个性化你的阅读

编程狂人

Programming Madman

NO.0

推酷

目录

做本电子版的技术周刊的想法由来已久了，现在终于迈出了第一步，有了我们第零期的《编程狂人》。期刊的内容是在已有的推荐数据基础上我们人工整理的，也努力呈现给大家值得阅读的内容。编程狂人们，走起！

业界新闻

应对全球性编程技能危机：从娃娃抓起

TIOBE 2013 年 11 月编程语言排行榜：C#、Transact-SQL 及 VB.NET 厚积薄发

Google 开发 Dart 语言已经完成，欢迎来到 Dart 1.0 的世界

兼具 SQL 与 NoSQL 特点：数据库初创企业 FoundationDB 获 1700 万美元融资

高效 Java Web 开发框架 JessMA v3.3.1 正式发布

比特币都有哪些基本概念和术语？如何通俗地理解它们？

目录

前端开发

关于网页排版和文章编辑那些事

构建自己的 AngularJS

1Kb 大小的 JavaScript 的 MVP 框架

网页重构应该避免的10大 CSS 糟糕用法

值得拥有！精心推荐几款超实用的 CSS 开发工具

CoffeeScript 尝鲜

编程语言

一个关于 Zend O+的小分享

Django 源码小剖: Django ORM 查询管理器 - 捣乱小子

在 Twitter，Netty 4 GC 开销降为五分之一

C++11 并发指南七(C++11 内存模型一：介绍) - Haippy

Meta Programming 主要解决什么了问题?

Qt 学习之路 2（70）：进程间通信

iOS 7 教程：浅析 Text Kit

目录

后端架构

RocksDB —来自 Facebook 的嵌入式 K/V 存储

NoSQL 数据库面面观

那些应对 APT 攻击的最新技术

[Lean Saas] MVP 的 M

NFS 挂了

Chef 在大数据集群部署中的应用

Cassandra 代替 Redis?

程序人生

编程能力与编程年龄

也谈谈程序员职业规划的几个问题

一个编程的自由人（图灵访谈）

COS 访谈第十三期：逄伟(eBay)

享受造轮子的乐趣

献给迷失的你——名 IT 员工的职场心得

应对全球性编程技能危机：从娃娃抓起

[**导读**]孩子在上学之前，就可以对科技技术有所掌握，东欧爱沙尼亚的做法，证明了这一点。



腾讯科技 晨曦 11月14日编译

在现实社会，程序员或者应用开发者，被视为“理工宅男”的典型，其工作和生活十分枯燥，并不是一个理想的职业。然而今天，移动设备和电脑渗透到了生活和工作的方方面面，程序员的稀缺堪称史无前例。那么，下一代的程序员从哪里来？目前，许多科技业者和国家，已经开始“从娃娃抓起”，让他们从小将编写程序作为一项适应计算时代的技能。

今天程序员的稀缺程度，堪比1980年代的超级模特。从银行到医院到政府部门，许多地方的 IT 部门，都需要程序员开发应用软件。不过，虽然越来越多的岗位需要编程技术，但是具备编程技能的人越来越少

显然，为了避免全球性的编程危机，社会需要帮助年轻人，让他们知道程序员并不是在一个黑屋子里吃着比萨饼、孑然一人工作的职业。

——机器人朋友

古普塔（Vikas-Gupta）曾经在谷歌(微博)担任负责支付业务的高管。在他看来，培养下一代程序员的关键，是从娃娃抓起，“我有一个两岁的女儿，想给孩子们做些事情，我开始思考，几岁大的孩子，可以掌握编程的概念？”

古普塔查询到麻省理工学院的一个研究结果，据称，只要通过现实互动手段学习，孩子们到了五岁，就可以掌握编程的概念。

“这个结论让我眼界大开，目前对这个年龄段的孩子们没有合适的（编程教育）产品，我决心设计一个产品，让十岁以下的孩子可以在娱乐中学习编程。”

他发明了两个小机器人“波”和“亚娜”，可以让孩子们在无意识中学习编程。“亚娜”个头小些，它可以根据程序发出声音或是做某种动作。“波”也可以移动。这两个小机器人还可以互动。

古普塔介绍说，通过编程，这两个机器人还可以玩“猫捉老鼠”的复杂游戏。一开始，孩子们可通过机器人玩很多的游戏，他们完全意识不到自己已经在编写程序。如果他们已经游刃有余，则可以转移到 Scratch 这样的可视化编程工具，编写自己人生的第一条代码。

这两个机器人可以通过平板或者智能手机进行无线控制，不过控制程序目前还只有苹果 iOS 版。“编程机器人”项目通过网络众贷方式融资，项目发布三天后，就融到了所期望的25万美元。

古普塔计划在明年夏天开始销售这种机器人，另外未来还要在学校中推广，他目前正在开发相关的教材资源。

——科技教育

在从上述项目的过程中，美国理工教育的现状让古普塔感到震惊，“在过去20年里，美国的计算机科学教育变得更糟糕，而不是更好。”

“美国计算机教师协会”2010年发布的报告，也证实了古普塔的观察。报告指出，美国三分之二的州在二级学校，只有很少或几乎没有计算机课程。另外，许多学校对于到底该教计算机科学还是计算机操作存在严重误解。

报告指出，在让学生获得未来所需的计算机知识和技术技能上，美国的学校已经远远落后。

——从第一天开始编程

孩子们在上学之前，就可以对科技技术有所掌握，东欧小国爱沙尼亚的做法，证明了这一点。

爱沙尼亚只有130万人口，但却拥有令人羡慕的电子政府计划，还诞生了 Skype 等知名科技公司。在科技方面，爱沙尼亚的做法领先其他大国。

爱沙尼亚的总统伊尔韦斯，曾经在大学时代靠编写代码补贴学费，他对于该国教育需要实现的目标十分清楚。这位总统获得了教育部长阿维克索（Jaak-Aaviksoo）的支持。

阿维克索曾经是一个物理学家，他认为，今天的教育，必须适应计算机时代，“19世纪延续下来的教育制度，是时候去适应21世纪带来的挑战了。”

爱沙尼亚的孩子，在七岁上小学之后，将开始学习编程课程。阿维克索表示：“孩子们很聪明，他们已经可以通过平板和智能手机完成复杂的事情，因此在学习编程时不会有很大距离。”

目前，小学编程课程还只是在试验阶段，阿维克索希望五年之内，推广到爱沙尼亚的所有小学。

——二次方程式

爱沙尼亚的教育改革，不仅仅是七岁学习编程这么简单。今年二月份，爱沙尼亚教育部宣布和英国牛津的 computerbasedmaths.org 组织展开合作。该组织负责人沃尔福莱姆（ConradWolfram）认为，目前数学教育的方式必须大幅改革，让孩子们进行复杂的计算没有意义，因为他们已经有了电脑，孩子们应该活用数学知识，以更加创新的方式，解决他们感兴趣的问题。

该组织认为，孩子们应该学会用数学解决实际问题，比如“怎样的形状是最美的？”、“我是否应该给我的笔记本电脑买一个保险？”、“我这个人是否正常？”

阿维克索正在考虑，是否让孩子们把自己的平板、电脑等设备带到课堂上。

在英格兰和威尔士等地，ICT（信息和通信技术）教育大纲也正在进行改革，按照英国教育部长 Michael-Gove 的设想，课程设置应该对今天的青少年更有吸引力，让他们为电脑主导的工作做好准备。

未来英国的学校将不再太看重考试成绩，而是重新强调传统的程序编写等动手能力。

原文链接： http://tech.qq.com/a/20131114/003000.htm

TIOBE 2013年11月编程语言排行榜：C#、Transact-SQL 及 VB.NET 厚积薄发

摘要：在前20名编程语言排行中，上升趋势较为明显的有3门语言是来自微软这个老东家开发的语言，它们分别是 C#、Transact-SQL 以及 Visual Basic.NET，这足以说明其流行程度也在日益增涨。另外，Go 语言成功挤进前50。

TIOBE 2013年11月份编程语言排行榜出炉，本月前三甲依然为：C、Java、Objective-

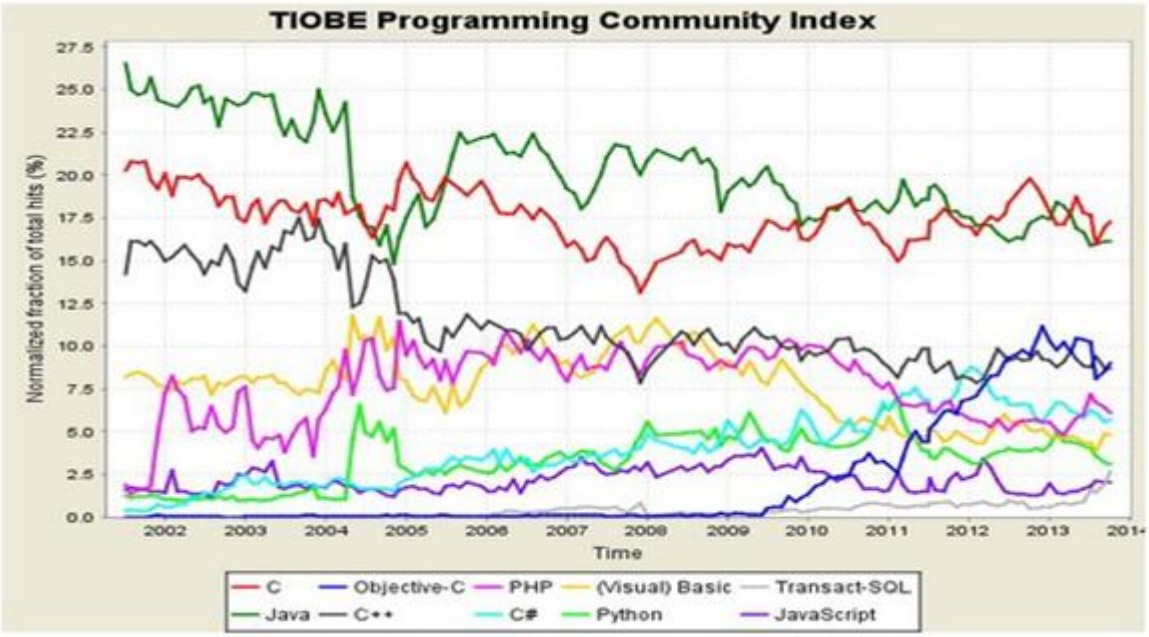
有趣的是，在前20名编程语言排行中，上升趋势较为明显的有3门语言是来自微软这个老东家开发的语言，这足以说明其流行程度也在日益增涨。它们分别是 C#、Transact-SQL 以及 Visual Basic.NET。在同一个月中出现这样的情况也许只是个巧合，但与 Android 和 iOS 手机操作系统相比，Windows Mobile 的市场份额的确有所上升

其他看点，JavaScript 呈现上升趋势，但不足以为奇，这是因为 JavaScript 是时下网站最流行的一门脚本语言之一。

另外值得一提的是，谷歌公司推出的 Go 语言（4岁了）从上个月的后50名排行榜中成功挤进前50，占据第49名位置，究竟 Go 能否崛起，让我们拭目以待。**以下是前20名榜单排行：**

Position Nov 2013	Position Nov 2012	Delta in Position	Programming Language	Ratings Nov 2013	Delta Nov 2012	Status
1	1	=	C	18.155%	-1.07%	A
2	2	=	Java	16.521%	-0.93%	A
3	3	=	Objective-C	9.406%	-0.98%	A
4	4	=	C++	8.369%	-1.33%	A
5	6	↑	C#	6.024%	+0.43%	A
6	5	↓	PHP	5.379%	-0.35%	A
7	7	=	(Visual) Basic	4.396%	-0.64%	A
8	8	=	Python	3.110%	-0.95%	A
9	23	↑↑↑↑↑↑↑↑↑↑	Transact-SQL	2.521%	+2.05%	A
10	11	↑	JavaScript	2.050%	+0.77%	A
11	15	↑↑↑↑	Visual Basic .NET	1.969%	+1.20%	A
12	9	↓↓↓	Perl	1.521%	-0.66%	A
13	10	↓↓↓	Ruby	1.303%	-0.44%	A
14	14	=	Pascal	0.715%	-0.17%	A
15	13	↓↓	Lisp	0.706%	-0.25%	A
16	19	↑↑↑	MATLAB	0.656%	+0.04%	B
17	12	↓↓↓↓↓	Delphi/Object Pascal	0.649%	-0.35%	A-
18	17	↓	PL/SQL	0.605%	-0.03%	A-
19	24	↑↑↑↑↑	COBOL	0.585%	+0.11%	B
20	20	=	Assembly	0.532%	-0.05%	B

前10名编程语言长期走势图：



以下是21-50编程语言排名：

Position	Programming Language	Ratings
21	SAS	0.519%
22	Ada	0.512%
23	F#	0.498%
24	Fortran	0.481%
25	Bash	0.475%
26	ABAP	0.458%
27	C shell	0.445%
28	Ladder Logic	0.435%
29	Logo	0.427%
30	Lua	0.425%
31	R	0.406%
32	Groovy	0.393%
33	D	0.392%
34	Common Lisp	0.387%
35	NXT-G	0.369%
36	Scheme	0.354%
37	Prolog	0.335%
38	Tcl	0.314%
39	PostScript	0.295%
40	VBScript	0.293%
41	Scala	0.292%
42	Erlang	0.272%
43	Scratch	0.269%
44	JavaFX Script	0.261%
45	PL/I	0.251%
46	Haskell	0.250%
47	ML	0.234%
48	RPG (OS/400)	0.224%
49	Go	0.214%
50	JScript.NET	0.212%

声明：这个榜单本身采集的是英文世界的数据，虽然在反映趋势上有一些参考意义，但与中国的实际情况不完全符合，而且，采样本身也有相当大的局限性。

【说明】TIOBE 编程语言社区排行榜是编程语言流行趋势的一个指标，每月更新，这份排行榜排名基于互联网上有经验的程序员、课程和第三方厂商的数量。排名使用著名的搜索引擎（诸如 Google、MSN、Yahoo!、Wikipedia、YouTube 以及 Baidu 等）进行计算。请注意这个排行榜只是反映某个编程语言的热门程度，并不能说明一门编程语言好不好，或者一门语言所编写的代码数量多少。

原文：http://www.csdn.net/article/2013-11-11/2817446-TIOBE

Google 开发 Dart 语言已经完成，欢迎来到 Dart 1.0 的世界



当然 Dart 只是“暂时”完成， Google 对其委以重任，日后一定会改进它。

来自 CNET 的报道称， Google 在比利时举行的 Devoxx 大会上发布了 Dart 语言的 1.0 版本。

两年前 Google 第一次宣布了 Dart 存在，这个互联网巨头不仅仅满足于使用流行的 JavaScript 语言，它看到了 JavaScript 可改进的地方，于是 Google 决定停止空谈，开始行动。

Google 设计 Dart 的目的是在 JavaScript 语言的基础上，改进编程效率和软件执行效率，并减少大型软件的编程复杂性。而从 Google 提供的 Benchmark 评分来看，它确实做到第一点， Dart 编写的程序执行效率比 JavaScript 高出 42%到 130%。



虽然愿景很好， Google 也确实做到了，但 Dart 的普及还困难重重。

像所有新生儿计算机语言面临的问题一样， Dart 没有丰富的库文件，而 JavaScript 经历这么多年发展，各种库文件已经不计其数，通过调用它们，程序员可以迅速、安全地完成目标程序。即将发布的 EcmaScript 6 也会给 JavaScript 带来性能改进，这让 Dart 性能方面的优势又小了一点。

抛开库文件不谈，目前只有很少的 Dart 程序员，各大浏览器开发商也不会针对 Dart 程序做什么优化，甚至根本不会引入 Dart 支持，原因很明显——没必要。 JavaScript 虽然有它的缺点，但毕竟已经在网络上风云这么多年，不计其数的网站以它做基础，突然引入 Dart 语言重写代码，需要大量的工作，而这些工作又是否有必要呢？



但 Google 对 Dart 非常有信心，而且并不着急推广。 Google 认为，目前网站程序的规模越来越大，别的不说，自家用 JavaScript 编写的 Google Docs，Google Sheets 和 Google Slides 就极度复杂，高度的复杂性也让管理维护它们变成一件非常头痛的事。而 Dart 正是为减少编程复杂性而设计，这一点也是 Google 非常向往的，在未来 Dart 的这个优势无疑将愈发突出。

Google 表示，公司已经开始在内部使用 Dart 语言，涉及领域包括客户关系管理、内部选举网站。除 Google 外，音乐录制网站 Soundtrap 、电邮公司 Mandrill 和相册公司 Montage 也在员工中推广 Dart 。Goolge 相信随着使用 Dart 的程序员越来越多，浏览器设计者也会开始重视起这个新生儿，慢慢地 Dart 就会普及。

目前 Google 推出了 Dart 开发工具包（ SDK ），有 JavaScript 经验的程序员学习起来会很快；鉴于当下的浏览器支持很不到位，Google 一并推出了 dart2js 工具，可以把 Dart 程序转换为网站可用的 JavaScript 程序。 Google 还发布了一款支持 Dart 的 Chrome 浏览器。

原文： http://www.ifanr.com/373944

兼具 SQL 与 NoSQL 特点：数据库初创企业 FoundationDB 获1700万美元融资



关系式数据库初创企业 FoundationDB 刚刚在 A 轮融资中获得 1700 万美元。

你一定奇怪为什么看似已经十分成熟的关系数据库技术怎么还会有人投资？其原因在于 FoundationDB 采用了与 Google 的 F1 类似的数据模型。后者是一种新型的可伸缩关系式数据库管理系统，Adwords 用的就是这种系统。与 F1 类似，FoundationDB 也具有可伸缩、可容错的特点，且能横跨多个分布式的基础设施之间而不会出现单点故障，而这个正是目前市面上大多数的 NoSQL 数据库固有的问题。此外，FoundationDB 的事务核心也是其相对优势，因为大多数的 NoSQL 数据库都还没有集成此类功能。这个事务核心功能就是 FoundationDB 所谓的“YesACID”—即具备确保数据库事务得到可靠处理的四性，原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)与持久性(Durability)。因此，可以说 FoundationDB 兼具了 SQL 和 NoSQL 的优点，可实现 MongoDB、SQL 及键值存储等数据库功能。该公司目前正开始将其产品开源供大众使用。

部署在分布式基础设施的数据库在进行跨数据中心的复制时往往会出现时延问题。这个问题 FoundationDB 是通过近乎无限的带宽和伸缩性来解决的，虽然单个复制可能花费的时间要久一点，但是系统总体的吞吐量是的查询可以以极高的速率并行进行。

可以说 FoundationDB 的技术令数据库类型不再是 SQL 或 NoSQL 那么的泾渭分明。这种混合了双方优点的新数据库技术大可适应于互联网规模的应用，小亦可满足小型团队所需的可靠性。也许这样的技术能在未来的数据库市场吸引到更多的客户。此轮融资由 Sutter Hill Ventures 领投，CrunchFund 及原有天使投资人跟投。

原文：http://www.36kr.com/p/207652.html

高效 Java Web 开发框架 JessMA v3.3.1 正式发布

JessMAMVCRESTWeb 开发框架 Java 开发框架



JessMA（原名：Portal-Basic）是一套功能完备的高性能 Full-Stack Web 应用开发框架，内置可扩展的 MVC Web 基础架构和 DAO 数据库访问组件（内部已提供了 Hibernate、MyBatis 与 JDBC DAO 组件），集成了 Action 拦截、Form / Dao / Spring Bean 装配、国际化、文件上传下载和页面静态化等基础 Web 应用组件，提供高效灵活的纯 Jsp/Servlet API 编程模型，可完美整合 Spring / Guice，支持 Action Convention，能快速开发传统风格和 RESTful 风格应用程序，文档和示例完善，非常容易学习。

JessMA 官方网站：http://www.jessma.org

JessMA 下载地址：http://www.oschina.net/p/portal-basic

JessMA 在线示例：http://demo.jessma.org

JessMA 在线 API 文档：http://www.jessma.org/doc

JessMA 在设计之初就充分注重功能、性能与使用体验。JessMA 主要特点：

★ 功能全面：内置稳定高效的 MVC 基础架构和 DAO 框架，支持 Action 拦截、Form Bean / Dao Bean / Spring Bean 装配和声明式事务，提供国际化、文件上传下载、缓存和页面静态化等常用 Web 组件，能满足绝大部分 Web 应用的需要。

★ 高度扩展：JessMA 通过的 plug-in 机制可以灵活扩展，JessMA 发布包中自带的 jessma-ext-rest 和 jessma-ext-spring 均以插件的形式提供，用户可根据需要加载或卸载这些插件。应用程序开发者也可以根据实际需要编写自定义插件来扩展 JessMA。

★ 强大的整合能力：JessMA 是一个 Full-Stack 框架，同时也是一个开放式框架，可以以非常简单的方式整合第三方组件。本开发手册会详细阐述如何在 JessMA 中整合 Freemarker、 Velocity 、 Urlrewrite、 EHCache-Web 、 Spring 、 Hibernate 和 Mybaits 等常用框架和组件。

★ 高性能：性能要求是 JessMA 的硬性指标，从每个模块的设计到每行代码的实现都力求简洁高效。另外，Portal-Basi c 并没有对 JSP/Servet API 进行过多封装，开发者仍然使用 JSP/Servet API 开发应用程序，没有过多的迂回，性能得到保证。

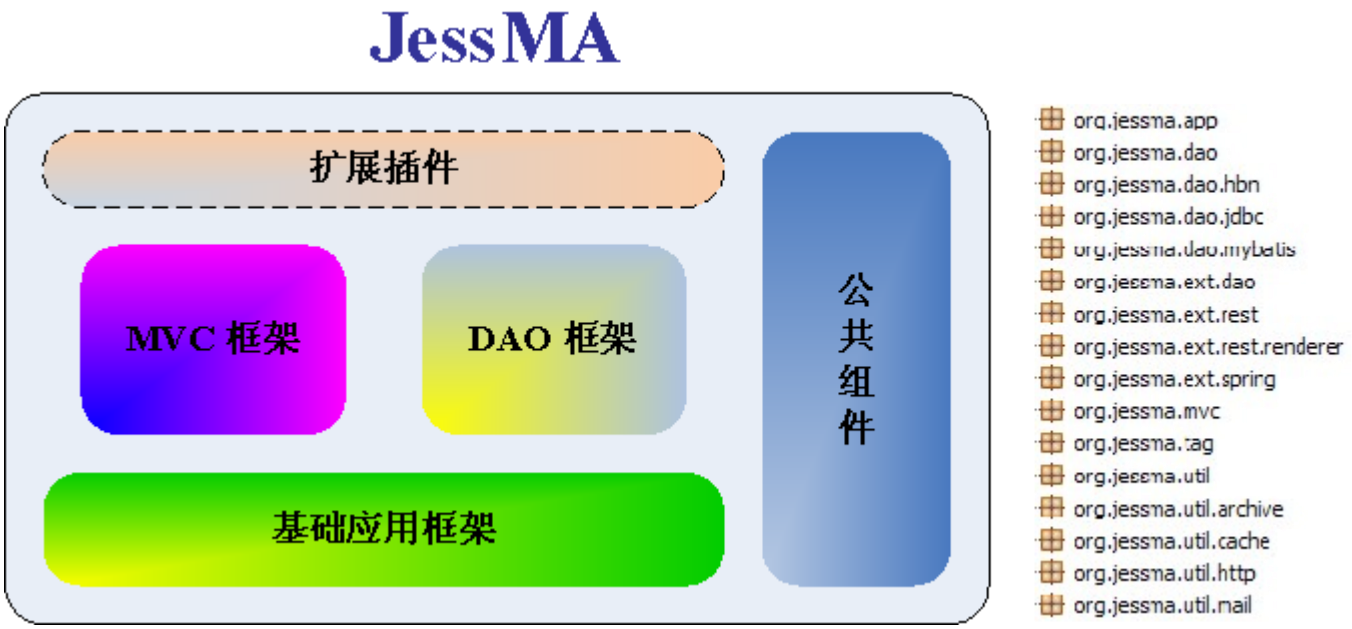
★ 优秀的使用体验：JessMA 的设计目标之一是提供良好的开发体验，尽量减少应用程序开发者的工作，API 的设计力求简单、完整、明确。同时，JessMA 为应用开发提供了大量 Util 工具，用来处理应用程序开发过程中通常会遇到的一般性问题，进一步减少应用程序开发者的工作负担。

★ 平缓的学习曲线：学习使用 JessMA 只需掌握一定的 Core Java 与 JSP/Servlet 知识，本开发手册会循序渐进阐述每个知识点，每个知识点都会结合完整的示例进行讲述，知识点之间前后呼应，确保学习者在学习时温故知新，融会贯通。

★ 完善的技术支持：除了提供完善的开发手册和示例代码以外，还提供博客和 QQ 群

用于解答使用 JessMA 过程中碰到的所有问题，也可以访问 JessMA 官方网站了解更多资讯。

*** JessMA 总体架构**



JessMA 主要包括以下5个部分：

- 基础应用框架

基础应用框架加载应用程序配置文件（默认：app-config.xml），监听应用程序的生命周期事件，并向上层应用发送应用程序启动和关闭通知，应用程序可以处理这些通知进行额外的初始化或清理工作。基础应用框架在 org.jessma.app 包中是实现。

- MVC 框架

MVC 框架加载 MVC 配置文件（默认：mvc-config.xml），通过前端控制器 ActionDispatcher 接收和解析所有的客户 HTTP 请求，然后交由相应的 Action 进行处理，最后生成相应的视图返回给客户端。MVC 框架在 org.jessma.mvc 包中实现。

● DAO 框架

DAO 框架封装了所有的数据库访问操作，内置 JDBC、Hibernate 和 MyBaits 数据库访问组件以及 Druid、Proxool、JNDI 等连接池。DAO 框架是可扩展的，用户可以通过扩展 org.jessma.dao.AbstractFacade 和 org.jessma.dao.AbstractSessionMgr 实现自己的数据库访问组件。DAO 框架在以下包中实现：

- org.jessma.dao
- org.jessma.dao.hbn
- org.jessma.dao.jdbc
- org.jessma.dao.mybatis

● 公共组件

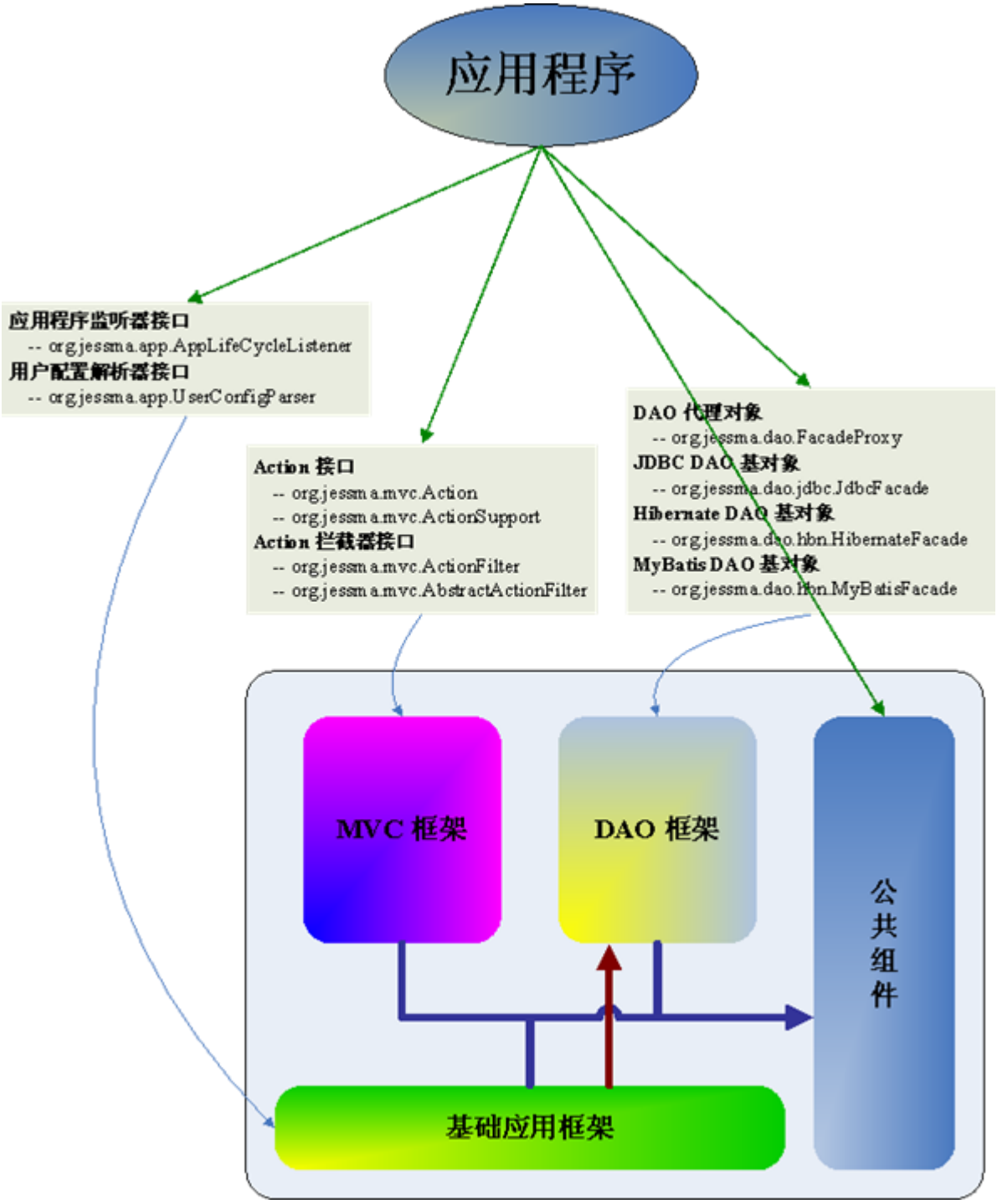
公共组件提供多种通用功能帮助类（如：字符串处理、类型转换、分页算法、压缩/解压、加解密、邮件发送等），这些类与框架无关，可在任何应用程序中使用。公共组件在以下包中实现：

- org.jessma.util
- org.jessma.util.archive
- org.jessma.util.cache
- org.jessma.util.http
- org.jessma.util.mail

● 扩展插件

扩展差插件是基于 JessMA 核心框架基础上的功能延伸，不是 JessMA 的必要组件。可由应用程序开发人员根据需要自行定制。JessMA 发行包中也自带了一些扩展插件（如：jessma-ext-spring 和 jessma-ext-rest）。

*** JessMA 应用程序依赖关系**



基础应用框架、MVC 框架和 DAO 框架都依赖于公共组件，其中基础应用框架同时依赖于 DAO 框架，因此，MVC 框架和 DAO 框架能脱离 JessMA 单独使用（当然，要附带上公共组件）。例如：可以把 MVC 框架和 DAO 框架用于 JessMA 之外的其它 Web 项目；也可以把 DAO 框架用于非 Web 项目（如：Swing / SWT 桌面应用）。

JessMA 3.3.1 更新：

1、增强国际化功能

1) 支持设置应用程序器默认 Locale 和默认 Bundle

A) 如果不配置器默认 Locale 则使用系统原来的默认 Locale

B) 如果不配置默认 Bundle，则默认 Bundle 为 'res.application-message'

（参考：下面的 MVC 配置文件条目）

```
<!--
    default-locale：应用程序默认语言选项（默认：与当前操作系统一致）
    default-bundle：应用程序默认 Bundle（默认：
res.application-message）
-->
<i18n default-locale="en_US"
default-bundle="com.bruce.res.application-message" />
```

2) 支持基于 URL、Cookie 或浏览器语言选项的国际化方式

A) org.jessma.mvc.i18n.URLI18nFilter :基于 URL 请求参数的国际化拦截器

B) org.jessma.mvc.i18n.CookieI18nFilter ：基于 Cookie 的国际化拦截器

C) org.jessma.mvc.i18n.BrowserI18nFilter：基于浏览器语言选项的国际化拦截器

D) JessMA 默认基于 Session 实现国际化,但可以通过配置上述拦截器改为相应的实现方式

E) 上述拦截器的国际化属性是基于 Request Attribute 的,因此会屏蔽 Session 的国际化设置

F) 上述拦截器可以组合使用，一般顺序为：URL -> Cookie -> Browser

（例如：下面同时配置了 URL、Cookie 和 Browser 三个国际化拦截器）

```
<action-filters>
    <!-- 优先检查客户请求中是否带有名称为 '__locale' 的请求参数 -->
    <filter class="org.jessma.mvc.i18n.URLI18nFilter"/>
    <!-- 然后检查请求是否带有名称为 '__locale' 的 Cookie -->
    <filter class="org.jessma.mvc.i18n.CookieI18nFilter"/>
    <!-- 最后根据客户端浏览器的语言选项设置当前请求的语言属性 -->
    <filter class="org.jessma.mvc.i18n.BrowserI18nFilter"/>
</action-filters>
```

2、支持 JSR 303 Bean Validation 机制

1) 如果开启了 Bean Validation 机制，应用程序可对由 @FormBean 注解的 Form Bean 执行自动验证

2) @FormBean 注解增加 'validate()' 和 'groups()' 两个属性，分别指示是否对 Form Bean 执行自动验证，以及自动验证的验证组

3) 如果自动验证成功则转入 Action 的 validate() 方法继续执行，否则会把错误信息写入 Action 的 errors 集合并跳转到 Action 的 INPUT 视图

4) 应用程序还可以随时调用 ActionSupport 的 validateBean() / validateBeanAndAddErrors() 方法验证其它任何 Bean

5) 增加错误信息输出标签 <p:err/> ,支持多种方式和格式展示错误信息(参考 :<p:err/> 及其实现类 org.jessma.tag.Error 相关注释)

（参考：下面的 MVC 配置文件条目）

```
<!--
    enable ：是否开启 Bean Validation 机制 ( 默认：开启 )
    bundle：验证文本消息的 Bundle ( 默认：res.validation-message )
    validator：验证器类 ( 默认：
org.jessma.mvc.validation.HibernateBeanValidator )
-->
<bean-validation
    enable="true"
    bundle="com.bruce.res.validation-message"
    validator="org.jessma.mvc.validation.HibernateBeanValidator"
/>
```

3、其它更新

1) org.jessma.ext.dao.* 包中的类移到 org.jessma.dao

2) 应用程序默认 Bundle 由 “res.message-resource” 改为 “res.application-message”

3) 修改 MVC 主/从配置文件、应用程序配置文件以及 REST 配置文件的 Sechema 引用（参考 MyJessMA 测试工程）

4) 更新开发手册《JessMA Java Web 应用开发框架 (v3.3.1)》

5) 整理依赖库 “jessma-lib”，删除多余的 jar 包4) 事务属性由最外层的 DAO 方法指定，忽略所有内层 DAO 方法的事务属性

4、应用程序升级说明

1) JessMA 3.2.x 升级到 JessMA 3.3.1 需要做以下工作：

A) 把原来对 org.jessma.ext.dao 包的引用改为 org.jessma.dao

B) 把资源文件 “res.message-resource” 改为 “res.application-message” 或在 <i18n/> 配置项中指定

C) 如有必要，修改 MVC 主从配置文件、应用程序配置文件以及 REST 配置文件的 Sechema

D) 如果要在 Tomcat-6.0 中开启 Bean Validation 机制，需要把 “tomcat_7_to_6_lib” 文件夹下的 3 个 jar 文件替换 Tomcat-6.0 的 lib 目录下的同名文件

2) Portal-Basic 3.1.1 及更早版本升级到 JessMA 3.3.1：（参考 JessMA 3.2.1 的升级说明）

JessMA 3.2.3 更新：

1、增加 Guice 支持

1) 增加 Guice 插件包 jessma-ext-guice-3.2.3.jar 用于整合 Guice

2) Action 通过 @GuiceBean/@GuiceBeans 注解声明 Guice Bean

3) Action 通过拦截器 com.bruce.ext.guice.GuiceInjectFilter 解析

@GuiceBean/@GuiceBeans 注解并注入 Guice Bean

4) 可以在 Guice Bean 中注入 JessMA DAO 对象，从而能在 Guice 环境下使用

JessMA DAO 子框架

5) 示例工程 MyJessMA 增加 Guice 整合示例

（注：通常情况下，应用程序需要创建 GuiceInjectFilter 的子类并改写

configModules() 方法，用于定义 Module 的绑定规则）

```
public class MyGuiceInjectFilter extends GuiceInjectFilter
{
    @Override
    protected Collection<Module> configModules()
    {
        Set<Module> modules = new HashSet<Module>();
        // 加入第一个 Module
        modules.add(new Module()
        {
            @Override
            public void configure(Binder binder)
            {
                // 配置绑定规则
                // ......
            }
        });
        // ......
        // 加入第N 个 Module
        modules.add(new Module()
        {
            @Override
```

```
            public void configure(Binder binder)
            {
                // 配置绑定规则

                // ......
            }
        });
        return modules;
    }
}
```

2、Form Bean 支持联级属性注入

1) @FormBean 注解支持注入联级属性

2) org.jessma.util.BeanHelper 的 createBean(...) /

setPropertiesOrFieldValues(...) 系列方法支持联级装配 Bean

（例如：下面的 Form 对应的 Bean 中，b、x 和 y 为联级 Bean）

```
<form>
    <input name="a" value="...">
    <input name="b.c" value="...">
    <input name="b.d" value="...">
    <input name="x.y.z" value="...">
</form>
```

3、规范“嵌套 DAO 方法”调用规则（详情参考：开发手册第 10.6 节）

1) 最外层的 DAO 对象必须由 FacadeProxy 创建

2) 内层的 DAO 对象可以用 new 或 FacadeProxy 等任意方式创建

3) 最外层的 DAO 方法作为一个事务单元，并且不会有嵌套事务

4) 事务属性由最外层的 DAO 方法指定，忽略所有内层 DAO 方法的事务属性

4、其它更新

1) 依赖库 “jessma-lib” 中的所有 jar 包更新到最新版本

2) 依赖库 “spring-lib” 中的所有 jar 包更新到最新版本

3) 增加 Guice 依赖库 “guice-lib”

4) 更新开发手册《JessMA Java Web 应用开发框架 (v3.2.3)》

5、应用程序升级说明

1) JessMA 3.2.2 升级到 JessMA 3.2.3：完全兼容，可直接升级

2) Portal-Basic 3.1.1 及更早版本升级到 JessMA 3.2.3：（参考 JessMA 3.2.1 的升级说明）

JessMA 3.2.2 更新：

(注：本次更新的主要内容是升级 DAO 组件)

1、org.jessma.dao.hbn.HibernateSessionMgr 支持自动扫描实体对象

1) 实体对象用 @Entity 注解取代 *.hbm.xml 映射文件（同时也不必在 hibernate.cfg.xml 中配置 ‘mapping’）

2) app-config.xml 中配置 HibernateSessionMgr 时，用第二个可选参数以正则表达式的格式指定实体对象所在包

（例如：实体对象位于 ‘com.bruce.<任意子包>.model’ 中）

```
    <manager name="mgr-1"
class="org.jessma.dao.hbn.HibernateSessionMgr">
        <initialize-args>
            <arg></arg>
            <arg>com\.bruce\..+\.model</arg>
        </initialize-args>
    </manager>
```

2、org.jessma.dao.mybatis.MyBatisSessionMgr 支持自动扫描 SQL Mapper 接口

1) 不必在 mybatis.cfg.xml 中配置 ‘mapper’

2) app-config.xml 中配置 MyBatisSessionMgr 时，用第三个可选参数以正则表达式的格式指定 SQL Mapper 接口所在包

（例如：SQL Mapper 接口位于 ‘com.bruce.<任意子包>.mapper’ 中）

```
    <manager name="mgr-2"
class="org.jessma.dao.mybatis.MyBatisSessionMgr">
        <initialize-args>
            <arg></arg>
            <arg></arg>
            <arg>com\.bruce\..+\.mapper</arg>
        </initialize-args>
    </manager>
```

3、所有 JDBC Session Manager 均取消 ‘isXml’ 配置参数，改为根据配置文件的扩展名自动识别配置文件类型

1) 受影响的 Session Manager：DruidSessionMgr、JdbcSessionMgr、JndiSessionMgr、ProxoolSessionMgr

4、org.jessma.dao.FacadeProxy 增加方法 executeCustomTransaction(...)

支持执行自定义事务

1) JessMA 的事务是 DAO 层事务，也就是说当外部调用某个 DAO 方法时，该方法作为一个事务单元执行。但在一些特殊情形下可能需要在 DAO 外部执行 Service 层事务(例如：事务需要调用多个 DAO 对象的多个方法）， 此时需要创建一个自定义事务（CustomTransaction），并调用 FacadeProxy 的 executeCustomTransaction(...) 来执行该自定义事务。

2) 增加自定义事务相关接口：

A) CustomTransaction ： 自定义事务基接口

B) JdbcTransaction ：JDBC 自定义事务接口

C) MyBatisTransaction ：MyBatis 自定义事务接口

D) HibernateTransaction：Hibernate 自定义事务接口

（示例）

```
public static void serviceMethod()
{
    // 获取 SessionMgr
    HibernateSessionMgr mgr =
(HibernateSessionMgr)AppConfig.getSessionManager("mgr-1");
    // 执行自定义事务
    FacadeProxy.executeCustomTransaction(mgr, new
HibernateTransaction(){
    // 实现自定义事务方法
        @Override
        public void execute(HibernateSessionMgr mgr) throws
DAOException
        {
```

```
            // 创建 dao1 （可以使用 FacadeProxy 创建 DAO 对象）
            MyDaoA dao1 = new MyDaoA(mgr);
            // 创建 dao2 （可以使用 ‘new MyDaoB(mgr)’ 创建 DAO 对象）
            MyDaoB dao2 = Facade.create(MyDaoB.class, mgr);

            // 执行 DAO 方法
            dao1.methodXxx();
            dao2.methodYyy();
            dao1.methodZzz();
            dao2.methodNnn();
        }});
    }
```

5、扩大 mvc-confing.xml 中 <result-path-aliases> 别名配置应用范围

1) 支持在 <result-path-aliases> 配置中引用前面定义的别名

（示例：下面配置中 ${index} 的实际路径为 '/jsp/test/index.jsp'）

```
<result-path-aliases>
    <alias name="jsp_base" path="/jsp/test"/>
    <alias name="index" path="${jsp_base}/index.jsp"/>
</result-path-aliases>
```

2) 允许在 <action-convention> 的 'dispatch-file-path' 和 'dispatch-file-path'

配置中使用别名

（示例：下面配置中 'dispatch-file-path' 的实际路径为 '/jsp'）

```
<action-convention
    dispatch-file-path="${ac_path}"
    <!-- 其它配置（略）... -->
/>
<result-path-aliases>
```

```
        <alias name="ac_path" path="/jsp"/>
    </result-path-aliases>
```

6、其它更新

1) 修改 org.jessma.util.PackageHelper，加入包扫描相关方法

2) 依赖库 “jessma-lib” 中的所有 jar 包更新到最新版本

3) 更新开发手册《JessMA 高效 Java MVC & REST 开发框架 (v3.2.2)》

4) 增加一个 JessMA 项目模板工程：HelloJessMA

7、应用程序升级说明

1) JessMA 3.2.1 升级到 JessMA 3.2.2：完全兼容，可直接升级

2) Portal-Basic 3.1.1 及更早版本升级到 JessMA 3.2.2：（参考 JessMA 3.2.1 的升级说明）

8、JessMA 3.2.1 历史更新：《JessMA 3.2.1 Release》

原文：http://blog.csdn.net/dumiyue/article/details/16818505

比特币都有哪些基本概念和术语？如何通俗地理解它们？

对于大部分人来说，比特币还是个比较新的概念，所以想请了解的人多多少少说一些比特币的术语和基本概念，方便普通人的理解。

1地址

比特币世界的地址大概长得像这样：

1EnvYExAb3JEkreirBufUhdeXSMyYyeNcC。

地址本身只是一串代码，就像你的银行账号3210000003288372一样，用来标示一个账户。

每个人可以有无数个地址。

地址是公开的。

比特币记录

银行账号里面有多少钱，是银行记录的。

比特币地址里面存有多少钱，是整个比特币网络共同记录的。每一份完整的比特币数据，都会记录着好1EnvYExAb3JEkreirBufUhdeXSMyYyeNcC 这个地址里面存了多少个比特币。

每个参与比特币网络的节点都可以保存一份完整的比特币数据，人人手头都有备份，防止造假。

密钥（私钥）

密钥是你自己才知道的另一串字符，密钥用来操作比特币地址里面的钱。

密钥和地址是一把钥匙一把锁的关系，每个地址只有一个密码，每个密码也操作一个比特币地址。

有了密钥以后，你就可以控制比特币地址里面的钱，支付给任何人。

密钥要绝对安全，丢失、删除的密钥永远找不回来，对应比特币地址里面的钱也就永远不能用了。

密钥需要绝对保密，任何知道密钥的人都可以偷走你的所有比特币。

钱包

比特币的交易过程会涉及很多复杂计算，为了简化操作，大家制作了钱包。

钱包有计算机上的客户端软件，也有在线网页版的。

钱包的功能是轻轻一点就可以付钱，背后的复杂计算交给钱包去完成。

既然要付钱，钱包当然要知道你的密钥才行，所以钱包另一个重要的功能就是保护密钥。

如果是客户端软件的话，密钥保存在软件设定的某个目录中（未加密）。如果是在线网页版的，密钥保存在网站后台。

有时候你还会听说“纸钱包”或者“脑钱包”的说法。纸钱包是说把你的密钥打印在一张纸上，然后从电脑里面彻底删除密钥记录。脑钱包则是把你自己方便记忆的一个词儿改成密钥，便于记忆。纸钱包和脑钱包都是保护密钥的一种方式。

安全性

如果使用计算机客户端软件的话，推荐 MultiBit，也可以用 Bitcoin-QT。这两个都是比特币社区的官方推荐软件。安装这两个软件一定只能从官方网站直接下载，下载前后一定要检查校验码。这么做是为了防止有人篡改钱包软件、盗窃密钥。

使用客户端软件作为钱包，密钥就保存在电脑中，所以电脑的安全也很重要。

如果使用纸钱包的话，务必保证纸张不要丢失。

不推荐脑钱包。

挖矿

大约每10分钟，比特币网络中都会新出现一些比特币，可以想象成天上在撒钱。

这些撒下来的钱落到谁的口袋里，主要看谁的电脑算得快。算得越快，就越有可能捡到这些钱。

挖矿就是用电脑拼命算，去捡这些钱。

普通家用电脑太慢了，早已经退出挖矿界。

挖矿的人俗称矿工。

刚刚接触的同志知道这些就差不多了。

背后的系统原理可以参看比特币 (Bitcoin) 系统是如何运行的。

原文链接：http://www.zhihu.com/question/22012510/answer/20021111

关于网页排版和文章编辑那些事

最近产品组的项目网站到了内网测试阶段。发现一些页面文章排版和编辑上的问题，这个对我们来说并不是什么新问题。简单说就是小编们在网站后台发布文章的时的一些问题。今天晚上无心码字，说说我的一些看法。

关于网页排版

有一说 Web Design is 95% Typography ，可见排版对于 web 设计来说有多么的重要。尤其是对于一些重视内容和阅读体验的网站（比如：简书、iA）就算没有复杂绚丽的交互、风格流行的色彩和图片，依然可以依靠舒适合理的排版布局而变得优秀。网页排版和传统平面排版虽然有一些基本的东西是相通的，但还是有一些不同。网页排版由于平台、终端、使用场景等的复杂性，所以不单纯是 web 设计师的事情，也包括前端重构狮，UED 甚至后端程序猿都需要牵扯进来。就不说什么多平台响应式设计、方块字和罗马字在排版上的不同处理，单是网页字体的使用，已经够写的一本书了。

段首缩进和段间距

段首缩进和段间距都是用来区分段落，明示逻辑关系，提高「**易读性**」的。其中**段首缩进是针对罗马字母系统所使用的排版方式，很多时候并不完全适用于方块字系统。在某些情况下会使得段首所空格数为非整数、不统一，影响阅读。**

很多小编并没有明白网页排版中段首**缩进的真正含义和用法**，这和传统出版行业还是不太一样的。小编们往往会认为首行缩进就是段落首行空两格，于是在每写一段之前都要啪啪键两下空格键。但是即使都是空格，全角和半角空格的占位字符还是不一样的。我想这主要是源于有些人从小写作文就段首空两格，于是觉得段首必须空两格。但完全没这回事。只要能在视觉上把段落隔开，就可以不用段首缩进。

段首缩进（尤其对于中文，缩进两字宽）只是分隔段落的手法之一。现代排版中还常用的另一个手法就是段间距。

段首缩进的优势是视觉效果低调，节省纵向空间，保障文本气息连贯。而段间距的优势是视觉效果显著，保障文本块整洁，提供阅读的气口。二者适合不同的场合。网页里用段间距能充分利用无限的纵向空间，并且在不怎么分页的长篇阅读体验中提供休息的间隙，避免长篇文字密密麻麻。**网页里段间距的确常常比段首缩进更合适。**

技术实现

首先需要理解的最重要的一点就是**内容和样式分离**，也就是说，编辑只管写，至于怎么缩进、超链接什么颜色、该用几号字号等等，这些还是交给 CSS 吧。从技术层面来说，只用可视化编辑器生成标签，不可以生成样式。总之，不要有空格和超过一次以上的回车就总是安全的。

段首缩进可以用 CSS 的 `text-indent:2em` 轻松实现，段间距一般设定 `<p>` 的纵向 `margin` 是最合理的方法。

关于段首缩进和段间距说了这么多了，这里推荐看一下知乎上知友 justjavac 的回答：网页文章编辑首行缩进有必要吗？

最后抄一段《老猫学出版》里的总结：

分段的目的是因为段落分明可以增加易读性，而不是因为有个标准的形式所以大家要遵守。报纸杂志只要空一格，就足以使段落分明，书版则常常需要空两格才能达到同样的效果，电脑屏幕呢？没错，在电脑上阅读，空行才有办法在视觉上产生同样的感受。
关于可视化编辑器

在特意使用后台的可视化编辑器编辑并上传了几篇文章之后，我失望地发现，后台的所见即所得编辑器功能太复杂且生成的代码很不干净；更别提改版之前的大量文章的格式没有统一，有的文章缩进了，有的没有用空格缩进，有的用半角空格缩进，有的用全角空格缩进。于是我狠下心找程序童鞋砍掉了大部分或许永远也用不到的可视化功能，之后编辑器才简单好用了一些，小伙伴们使用后都大呼畅快。

作为前端小狮子和自认对网页排版相对有一些认识的我，给网站的（主要是）编辑童鞋编写了一份**网页排版和编辑规范**的文档。包括告诉他们什么是网页的内容和样式的分离，后台发文章的时候不要带任何的样式，包括段间空行，段首缩进，告诉他们这些是 CSS 的任务不是编辑的工作。他们只管写得优质的内容，网页排版和样式上，做得越少，就是做得越好（*less is more*）。

我认为可视化编辑器对于 HTML5时代的网页排版设计来说已经开始过时了。HTML 代码已经非常简单，并且 HTML5新增加了一些非常好的语义化标签，即使没有一点基础的编辑童鞋，花半天时间也能了解个差不多。其实网页上文章内容最常用的标签无外乎就那十几个。

如果 HTML“代码”会让你觉得心生畏惧的话，那么我强烈推荐你学一下 Markdown，它是一种轻量级标记语言，它的语法非常简单，甚至不能算得上是一种计算机语言。而只是一些简单的有语法意义的符号。他还可以方便的生成 HTML 格式的文档。关于 Markdown 的用法在此就不多说了，推荐阅读 开始使用 Markdown 、 简书的 Markdown 专栏 ，想完整的学习请转 Markdown 语法说明。试试吧，你会爱上它的。

关于网页字体

字体是一门很深的学问，网页字体的运用更是复杂。在此我不敢做更多的谈论。只简单说说我自己理解的几点：

同样的字体在不同操作系统、浏览器下的最终显示效果都可能有不一样

CSS3 的 web-font 属性的出现，使得网页字体比以往任何时候得到 web 设计师的青睐，但不包括中文字体

个人觉得 Mac 上的字体显示真的比 Windows 上的字体好看太多（主要是渲染技术上的不同

（非字体方面）关于字号的设置，推荐 REM 设置文字大小。个人觉得网页上12像素字体太小了，14px 是底线，16px 刚好。现在网页设计趋向于用更大的字号，我想其中一个原因就是屏幕分辨率越来越高了吧。

想了解字体的内容，个人非常推荐下面的文章：

字体是网页设计中最重要的细节](http://www.qianxingzhem.com/post-1588.html

字体渲染详解](http://www.21haolou.com/articles/show/153

认识字体渲染](认识字体渲染

最后的最后，我的推荐强迫症

推荐网站：

简书 （要内容有内容，要排版有排版，要 Markdown 有 Markdown

iA （英文排版的典范。简洁，极致）

推荐我读过的两本书：

《简约之上 - 交互式设计四策略》（你难道不喜欢简单易用的产品吗？

《写给大家看的设计书》（设计菜鸟、产品、前端、网页编辑都应该读一读）

原文：

http://huangyang.me/post/about-web-typography.html?utm_source=Tuicool_Weekly

构建自己的 AngularJS，第一部分：Scope 和 Digest

Angular 是一个成熟和强大的 JavaScript 框架。它也是一个比较庞大的框架，在熟练掌握之前，需要领会它提出的很多新概念。很多 Web 开发人员涌向 Angular，有不少人面临同样的障碍。Digest 到底是怎么做的？定义一个指令（directive）有哪些不同的方法？Service 和 provider 有什么区别？

Angular 的文档挺不错的，第三方的资源也越来越丰富，想要学习一门新的技术，没什么方法比把它拆开研究其运作机制更好。

在这个系列的文章中，我将从无到有构建 AngularJS 的一个实现。随着逐步深入的讲解，读者将能对 Angular 的运作机制有一个深入的认识。

在第一部分中，读者将看到 Angular 的作用域是如何运作的，还有比如$eval, $digest, $apply 这些东西怎么实现。Angular 的脏检查逻辑看上去有些不可思议，但你将看到实际并非如此。

基础知识

在 Github 上，可以看到这个项目的全部源码。相比只复制一份下来，我更建议读者从无到有构建自己的实现，从不同角度探索代码的每个步骤。在本文中，我嵌入了 JSBin 的一些代码，可以直接在文章中进行一些互动。（译者注：因为我在 github 上翻译，没法集成 JSBin 了，只能给链接……）

我们将使用 Lo-Dash 库来处理一些在数组和对象上的底层操作。Angular 自身并未使用 Lo-Dash，但是从我们的目的看，要尽量无视这些不太相关的比较底层的事情。当读者在代码中看到下划线（_）的时候，那就是在调用 Lo-Dash 的功能。

我们还将使用 console.assert 函数做一些特别的测试。这个函数应该适用于所有现代 JavaScript 环境。

下面是使用 Lo-Dash 和 assert 函数的示例：

http://jsbin.com/UGOVUk/4/embed?js,console

Scope 对象

Angular 的 Scope 对象是 POJO（简单的 JavaScript 对象），在它们上面，可以像对其他对象一样添加属性。Scope 对象是用构造函数创建的，我们来写个最简单的版本：

```
function Scope() {}
```

现在我们就可以使用 new 操作符来创建一个 Scope 对象了。我们也可以在它上面附加一些属性：

```
var aScope = new Scope();aScope.firstName = 'Jane';aScope.lastName = 'Smith';
```

这些属性没什么特别的。不需要调用特别的设置器（setter），赋值的时候也没什么限制。相反，在两个特别的函数：$watch 和$digest 之中发生了一些奇妙的事情。

监控对象属性：$watch 和$digest

$watch 和$digest 是相辅相成的。两者一起，构成了 Angular 作用域的核心：数据变化的响应。

使用$watch，可以在 Scope 上添加一个监听器。当 Scope 上发生变更时，监听器会收到提示。给$watch 指定如下两个函数，就可以创建一个监听器：

24 一个监控函数，用于指定所关注的那部分数据。
25 一个监听函数，用于在数据变更的时候接受提示。

作为一名 Angular 用户，一般来说，是监控一个表达式，而不是使用监控函数。监控表达式是一个字符串，比如说“user.firstName”，通常在数据绑定，指令的属性，或者 JavaScript 代码中指定，它被 Angular 解析和编译成一个监控函数。在这篇文章的后面部分我们会探讨这是如何做的。在这篇文章中，我们将使用稍微低级的方法直接提供监控功能。

为了实现$watch，我们需要存储注册过的所有监听器。我们在 Scope 构造函数上添加一个数组：

```
function Scope() { this.$$watchers = [];}
```

在 Angular 框架中，双美元符前缀$$表示这个变量被当作私有的来考虑，不应当在外部代码中调用。

现在我们可以定义$watch 方法了。它接受两个函数作参数，把它们存储在$$watchers 数组中。我们需要在每个 Scope 实例上存储这些函数，所以要把它放在 Scope 的原型上：

```
Scope.prototype.$watch = function(watchFn, listenerFn) { var watcher = { watchFn: watchFn, listenerFn: listenerFn
  }; this.$$watchers.push(watcher);};
```

另外一面就是$digest 函数。它执行了所有在作用域上注册过的监听器。我们来实现一个它的简化版，遍历所有监听器，调用它们的监听函数：

```
Scope.prototype.$digest = function() { _.forEach(this.$$watchers, function(watch) { watch.listenerFn(); });
};
```

现在我们可以添加监听器，然后运行$digest 了，这将会调用监听函数：

http://jsbin.com/oMaQoxa/2/embed?js,console

这些本身没什么大用，我们要的是能检测由监控函数指定的值是否确实变更了，然后调用监听函数。

脏值检测

如同上文所述，监听器的监听函数应当返回我们所关注的那部分数据的变化，通常，这部分数据就存在于作用域中。为了使得访问作用域更便利，在调用监控函数的时候，使用当前作用域作为实参。一个关注作用域上 fiestName 属性的监听器像这个样子：

```
function(scope) {    return  scope.firstName;}
```

这是监控函数的一般形式：从作用域获取一些值，然后返回。

$digest 函数的作用是调用这个监控函数，并且比较它返回的值和上一次返回值的差异。如果不相同，监听器就是脏的，它的监听函数就应当被调用。

想要这么做，$digest 需要记住每个监控函数上次返回的值。既然我们现在已经为每个监听器创建过一个对象，只要把上一次的值存在这上面就行了。下面是检测每个监控函数值变更的$digest 新实现：

```
Scope.prototype.$digest = function() {    var self = this;    _.forEach(this.$$watchers, function(watch) {        var newValue = watch.watchFn(self);        var oldValue = watch.last;        if (newValue !== oldValue) {            watch.listenerFn(newValue, oldValue, self);        }        watch.last = newValue;    });
};
```

对每个监听器，我们调用监控函数，把作用域自身当作实参传递进去，然后比较这个返回值和上次返回值，如果不同，就调用监听函数。方便起见，我们把新旧值和作用域都当作参数传递给监听函数。最终，我们把监听器的 last 属性设置成新返回的值，下一次可以用它来作比较。

有了这个实现之后，我们就可以看到在$digest 调用的时候，监听函数是怎么执行的：

http://jsbin.com/OsITIZu/3/embed?js,console

我们已经实现了 Angular 作用域的本质：添加监听器，在 digest 里运行它们。

也已经可以看到几个关于 Angular 作用域的重要性能特性：

在作用域上添加数据本身并不会有性能折扣。如果没有监听器在监控某个属性，它在不在作用域上都无所谓。Angular 并不会遍历作用域的属性，它遍历的是监听器。

$digest 里会调用每个监控函数，因此，最好关注监听器的数量，还有每个独立的监控函数或者表达式的性能。

在 Digest 的时候获得提示

如果你想在每次 Angular 的作用域被 digest 的时候得到通知，可以利用每次 digest 的时候挨个执行监听器这个事情，只要注册一个没有监听函数的监听器就可以了。

想要支持这个用例，我们需要在$watch 里面检测是否监控函数被省略了，如果是这样，用个空函数来代替它：

```
Scope.prototype.$watch = function(watchFn, listenerFn) {  var watcher = {  watchFn: watchFn,  listenerFn: listenerFn || function() { }  };  this.$$watchers.push(watcher);};
```

如果用了这个模式，需要记住，即使没有 listenerFn，Angular 也会寻找 watchFn 的返回值。如果返回了一个值，这个值会提交给脏检查。想要采用这个用法又想避免多余的事情，只要监控函数不返回任何值就行了。在这个例子里，监听器的值始终会是未定义的。

http://jsbin.com/OsITIZu/4/embed?js,console

这个实现的核心就这样，但是离最终的还是差太远了。比如说有个很典型的场景我们不能支持：监听函数自身也修改作用域上的属性。如果这个发生了，另外有个监听器在监控被修改的属性，有可能在同一个 digest 里面检测不到这个变动：

http://jsbin.com/eTIpUyE/2/embed?js,console

我们来修复这个问题。

当数据脏的时候持续 Digest

我们需要改变一下 digest，让它持续遍历所有监听器，直到监控的值停止变更。

首先，我们把现在的$digest 函数改名为$$digestOnce，它把所有的监听器运行一次，返回一个布尔值，表示是否还有变更了：

```
Scope.prototype.$$digestOnce = function() {  var self  = this;  var dirty;  _.forEach(this.$$watchers, function(watch) {  var newValue = watch.watchFn(self);  var oldValue = watch.last;  if (newValue !== oldValue) {  watch.listenerFn(newValue, oldValue, self);  dirty = true;  }  watch.last = newValue;  });  return dirty;};
```

然后，我们重新定义$digest，它作为一个“外层循环”来运行，当有变更发生的时候，调用$$digestOnce：

```
Scope.prototype.$digest = function() {  var dirty;  do {  dirty = this.$$digestOnce();  } while (dirty);};
```

$digest 现在至少运行每个监听器一次了。如果第一次运行完，有监控值发生变更了，标记为 dirty，所有监听器再运行第二次。这会一直运行，直到所有监控的值都不再变化，整个局面稳定下来了。

Angular 作用域里并不是真的有个函数叫做$$digestOnce，相反，digest 循环都是包含在$digest 里的。我们的目标更多是清晰度而不是性能，所以把内层循环封装成了一个函数。

下面是新的实现：

http://jsbin.com/Imoyosa/3/embed?js,console

我们现在可以对 Angular 的监听器有另外一个重要认识：它们可能在单次 digest 里面被执行多次。这也就是为什么人们经常说，监听器应当是幂等的：一个监听器应当没有边界效应，或者边界效应只应当发生有限次。比如说，假设一个监控函数触发了一个 Ajax 请求，无法确定你的应用程序发了多少个请求。

在我们现在的实现中，有一个明显的遗漏：如果两个监听器互相监控了对方产生的变更，会怎样？也就是说，如果状态始终不会稳定？这种情况展示在下面的代码里。在这个例子里，$digest 调用被注释掉了，把注释去掉看看发生什么情况：

http://jsbin.com/eKEvOYa/3/embed?js,console

JSBin 执行了一段时间之后就停止了（在我机器上大概跑了100,000次左右）。如果你在别的东西比如 Node.js 里跑，它会一直运行下去。

放弃不稳定的 digest

我们要做的事情是，把 digest 的运行控制在一个可接受的迭代数量内。如果这么多次之后，作用域还在变更，就勇敢放手，宣布它永远不会稳定。在这个点上，我们会抛出一个异常，因为不管作用域的状态变成怎样，它都不太可能是用户想要的结果。

迭代的最大值称为 TTL（short for Time To Live）。这个值默认是10，可能有点小（我们刚运行了这个 digest 100,000次！），但是记住这是一个性能敏感的地方，因为 digest 经常被执行，而且每个 digest 运行了所有的监听器。用户也不太可能创建10个以上链状的监听器。

事实上，Angular 里面的 TTL 是可以调整的。我们将在后续文章讨论 provider 和依赖注入的时候再回顾这个话题。

我们继续，给外层 digest 循环添加一个循环计数器。如果达到了 TTL，就抛出异常：

```
Scope.prototype.$digest = function() {  var ttl = 10;  var dirty;  do {  dirty = this.$$digestOnce();  if (dirty && !(ttl--)) {  throw "10 digest iterations reached";  }  } while (dirty);};
```

下面是更新过的版本，可以让我们循环引用的监控例子抛出异常：

http://jsbin.com/uNapUWe/2/embed?js,console

这些应当已经把 digest 的事情说清楚了。

现在，我们把注意力转到如何检测变更上吧。

基于值的脏检查

我们曾经使用严格等于操作符(===)来比较新旧值，在绝大多数情况下，它是不错的，比如所有的基本类型（数字，字符串等等），也可以检测一个对象或者数组是否变成新的了，但 Angular 还有一种办法来检测变更，用于检测当对象或者数组内部产生变更的时候。那就是：可以监控值的变更，而不是引用。

这类脏检查需要给$watch 函数传入第三个布尔类型的可选参数当标志来开启。当这个标志为真的时候，基于值的检查开启。我们来重新定义$watch，接受这个参数，并且把它存在监听器里：

```
Scope.prototype.$watch = function(watchFn, listenerFn, valueEq) { var watcher = { watchFn: watchFn, listenerFn: listenerFn, valueEq: !!valueEq
  }; this.$$watchers.push(watcher);};
```

我们所做的一切是把这个标志加在监听器上，通过两次取反，强制转换为布尔类型。当用户调用$watch，没传入第三个参数的时候，valueEq 会是未定义的，在监听器对象里就变成了 false。

基于值的脏检查意味着如果新旧值是对象或者数组，我们必须遍历其中包含的所有内容。如果它们之间有任何差异，监听器就脏了。如果该值包含嵌套的对象或者数组，它也会递归地按值比较。

Angular 内置了自己的相等检测函数，但是我们会用 Lo-Dash 提供的那个。让我们定义一个新函数，取两个值和一个布尔标志，并比较相应的值：

```
Scope.prototype.$$areEqual = function(newValue, oldValue, valueEq) {  if (valueEq) {  return _.isEqual(newValue, oldValue);  } else {  return newValue === oldValue;  }};
```

为了提示值的变化，我们也需要改变之前在每个监听器上存储旧值的方式。只存储当前值的引用是不够的，因为在这个值内部发生的变更也会生效到它的引用上，$$areEqual 方法比较同一个值的两个引用始终为真，监控不到变化，因此，我们需要建立当前值的深拷贝，并且把它们储存起来。

就像相等检测一样，Angular 也内置了自己的深拷贝函数，但我们还是用 Lo-Dash 提供的。我们修改一下$digestOnce，在内部使用新的$$areEqual 函数，如果需要的话，也复制最后一次的引用：

```
Scope.prototype.$$digestOnce = function() {  var self  = this;  var dirty;  _.forEach(this.$$watchers, function(watch) {  var newValue = watch.watchFn(self);  var oldValue = watch.last;  if (!self.$$areEqual(newValue, oldValue, watch.valueEq)) {  watch.listenerFn(newValue, oldValue, self);  dirty = true;  }  watch.last = (watch.valueEq ? _.cloneDeep(newValue) : newValue);  });  return dirty;};
```

现在我们可以看到两种脏检测方式的差异：

http://jsbin.com/ARiWENO/3/embed?js,console

相比检查引用，检查值的方式显然是一个更为复杂的操作。遍历嵌套的数据结构很花时间，保持深拷贝的数据也占用不少内存。这就是 Angular 默认不使用基于值的脏检测的原因，用户需要显式设置这个标记去打开它。

Angular 也提供了第三种脏检测的方法：集合监控。就像基于值的检测，也能提示对象和数组中的变更。但不同于基于值的检测方式，它做的是一个比较浅的检测，并不递归进入到深层去，所以它比基于值的检测效率更高。集合检测是通过“$watchCollection”函数来使用的，在这个系列的后续部分，我们会来看看它是如何实现的。

在我们完成值的比对之前，还有些 JavaScript 怪事要处理一下。

非数字（NaN）

在 JavaScript 里，NaN（Not-a-Number）并不等于自身，这个听起来有点怪，但确实就这样。

如果我们在脏检测函数里不显式处理 NaN，一个值为 NaN 的监听器会一直是脏的。

对于基于值的脏检测来说，这个事情已经被 Lo-Dash 的 isEqual 函数处理掉了。对于基于引用的脏检测来说，我们需要自己处理。来修改一下$$areEqual 函数的代码：

```
Scope.prototype.$$areEqual = function(newValue, oldValue, valueEq) {  if (valueEq) {
return _.isEqual(newValue, oldValue);  } else {      return newValue === oldValue ||
 (typeof newValue === 'number' && typeof oldValue === 'number' &&          isNaN(new
Value) && isNaN(oldValue));  }};
```

现在有 NaN 的监听器也正常了：

http://jsbin.com/ijINaRA/2/embed?js,console

基于值的检测实现好了，现在我们该把注意力集中到应用程序代码如何跟作用域打交道上了。

$eval - 在作用域的上下文上执行代码

在 Angular 中，有几种方式可以在作用域的上下文上执行代码，最简单的一种就是$eval。它使用一个函数作参数，所做的事情是立即执行这个传入的函数，并且把作用域自身当作参数传递给它，返回的是这个函数的返回值。$eval 也可以有第二个参数，它所做的仅仅是把这个参数传递给这个函数。

$eval 的实现很简单：

```
Scope.prototype.$eval = function(expr, locals) {  return expr(this, locals);};
```

$eval 的使用一样很简单：

http://jsbin.com/UzaWUC/1/embed?js,console

那么，为什么要用这么一种明显很多余的方式去执行一个函数呢？有人觉得，有些代码是专门与作用域的内容打交道的，$eval 让这一切更加明显。$scope 也是构建$apply 的一个部分，后面我们就来讲它。

然后，可能$eval 最有意思的用法是当我们不传入函数，而是表达式。就像$watch 一样，可以给$eval 一个字符串表达式，它会把这个表达式编译，然后在作用域的上下文中执行。我们将在这个系列的后面部分实现这些。

$apply - 集成外部代码与 digest 循环

可能 Scope 上所有函数里最有名的就是$apply 了。它被誉为将外部库集成到 Angular 的最标准的方式，这话有个不错的理由。

$apply 使用函数作参数，它用$eval 执行这个函数，然后通过$digest 触发 digest 循环。下面是一个简单的实现：

```
Scope.prototype.$apply = function(expr) {    try {        return this.$eval(expr);    } finally {
  this.$digest();    }};
```

$digest 的调用放置于 finally 块中，以确保即使函数抛出异常，也会执行 digest。

关于$apply，大的想法是，我们可以执行一些与 Angular 无关的代码，这些代码也还是可以改变作用域上的东西，$apply 可以保证作用域上的监听器可以检测这些变更。当人们谈论使用$apply 集成代码到“Angular 生命周期”的时候，他们指的就是这个事情，也没什么比这更重要的了。

这里是$apply 的实践：

http://jsbin.com/UzaWUC/2/embed?js,console

延迟执行 - $evalAsync

在 JavaScript 中，经常会有把一段代码“延迟”执行的情况 - 把它的执行延迟到当前的执行上下文结束之后的未来某个时间点。最常见的方式就是调用 setTimeout()函数，传递一个0（或者非常小）作为延迟参数。

这种模式也适用于 Angular 程序，但更推荐的方式是使用$timeout 服务，并且使用$apply 把要延迟执行的函数集成到 digest 生命周期。

但在 Angular 中还有一种延迟代码的方式，那就是 Scope 上的$evalAsync 函数。$evalAsync 接受一个函数，把它列入计划，在当前正持续的 digest 中或者下一次 digest 之前执行。举例来说，你可以在一个监听器的监听函数中延迟执行一些代码，即使它已经被延迟了，仍然会在现有的 digest 遍历中被执行。

我们首先需要的是存储$evalAsync 列入计划的任务，可以在 Scope 构造函数中初始化一个数组来做这事：

```
function Scope() {  this.$$watchers = [];  this.$$asyncQueue = [];}
```

我们再来定义$evalAsync，它添加将在这个队列上执行的函数：

```
Scope.prototype.$evalAsync = function(expr) {  this.$$asyncQueue.push({scope: this, expression: expr});};
```

我们显式在放入队列的对象上设置当前作用域，是为了使用作用域的继承，在这个系列的下一篇文章中，我们会讨论这个。

然后，我们在$digest 中要做的第一件事就是从队列中取出每个东西，然后使用$eval 来触发所有被延迟执行的函数：

```
Scope.prototype.$digest = function() {  var ttl = 10;  var dirty;  do {  while (this.$$asyncQueue.length) {  var asyncTask = this.$$asyncQueue.shift();  this.$eval(asyncTask.expression);  }  dirty = this.$$digestOnce();  if (dirty && !(ttl--)) {  throw "10 digest iterations reached";  }  } while (dirty);};
```

这个实现保证了：如果当作用域还是脏的，就想把一个函数延迟执行，那这个函数会在稍后执行，但还处于同一个 digest 中。

下面是关于如何使用$evalAsync 的一个示例：

http://jsbin.com/ilepOwI/1/embed?js,console

作用域阶段

$evalAsync 做的另外一件事情是：如果现在没有其他的$digest 在运行的话，把给定的$digest 延迟执行。这意味着，无论什么时候调用$evalAsync，可以确定要延迟执行的这个函数会“很快”被执行，而不是等到其他什么东西来触发一次 digest。

需要有一种机制让$evalAsync 来检测某个$digest 是否已经在运行了，因为它不想影响到被列入计划将要执行的那个。为此，Angular 的作用域实现了一种叫做阶段（phase）的东西，它就是作用域上一个简单的字符串属性，存储了现在正在做的信息。

在 Scope 的构造函数里，我们引入一个叫$$phase 的字段，初始化为 null：

```
function Scope() {  this.$$watchers = [];  this.$$asyncQueue = [];  this.$$phase = null;}
```

然后，我们定义一些方法用于控制这个阶段变量：一个用于设置，一个用于清除，也加个额外的检测，以确保不会把已经激活状态的阶段再设置一次：

```
Scope.prototype.$beginPhase = function(phase) {  if (this.$$phase) {  throw this.$$phase + ' already in progress.';  }  this.$$phase = phase;};
```

```
Scope.prototype.$clearPhase = function() {  this.$$phase = null;};
```

在$digest 方法里，我们来从外层循环设置阶段属性为“$digest”：

```
Scope.prototype.$digest = function() {  var ttl = 10;  var dirty;  this.$beginPhase("$digest");  do {  while (this.$$asyncQueue.length) {  var asyncTask = this.$$asyncQueue.shift();  this.$eval(asyncTask.expression);  }  dirty = this.$$digestOnce();  if (dirty && !(ttl--)) {  this.$clearPhase();  throw "10 digest iterations reached";  }  } while (dirty);  this.$clearPhase();};
```

我们把$apply 也修改一下，在它里面也设置个跟自己一样的阶段。在调试的时候，这个会有些用：

```
Scope.prototype.$apply = function(expr) {  try {  this.$beginPhase("$apply");  return this.$eval(expr);  } finally {  this.$clearPhase();  this.$digest();  }};
```

最终，把对$digest 的调度放进$evalAsync。它会检测作用域上现有的阶段变量，如果没有（也没有已列入计划的异步任务），就把这个 digest 列入计划。

```
Scope.prototype.$evalAsync = function(expr) {  var self = this;  if (!self.$$phase && !self.$$asyncQueue.length) {  setTimeout(function() {  if (self.$$asyncQueue.length) {  self.$digest();  }  }, 0);  }  self.$$asyncQueue.push({scope: self, expression: expr});};
```

有了这个实现之后，不管何时、何地，调用$evalAsync，都可以确定有一个 digest 会在不远的将来发生。

http://jsbin.com/iKeSaGi/1/embed?js,console

在 digest 之后执行代码 - $$postDigest

还有一种方式可以把代码附加到 digest 循环中，那就是把一个$$postDigest 函数列入计划。

在 Angular 中，函数名字前面有双美元符号表示它是一个内部的东西，不是应用开发人员应该用的。但它确实存在，所以我们也要把它实现出来。

就像$evalAsync 一样，$$postDigest 也能把一个函数列入计划，让它“以后”运行。具体来说，这个函数将在下一次 digest 完成之后运行。将一个$$postDigest 函数列入计划不会导致一个 digest 也被延后，所以这个函数的执行会被推迟到直到某些其他原因引起一次 digest。顾名思义，$$postDigest 函数是在 digest 之后运行的，如果你在$$digest 里面修改了作用域，需要手动调用$digest 或者$apply，以确保这些变更生效。

首先，我们给 Scope 的构造函数加队列，这个队列给$$postDigest 函数用：

```
function Scope() { this.$$watchers = []; this.$$asyncQueue = []; this.$$postDigestQueue = []; this.$$phase = null;}
```

然后，我们把$$postDigest 也加上去，它所做的就是把给定的函数加到队列里：

```
Scope.prototype.$$postDigest = function(fn) { this.$$postDigestQueue.push(fn);};
```

最终，在$digest 里，当 digest 完成之后，就把队列里面的函数都执行掉。

```
Scope.prototype.$digest = function() { var ttl = 10; var dirty; this.$beginPhase("$digest"); do { while (this.$$asyncQueue.length) { var asyncTask = this.$$asyncQueue.shift(); this.$eval(asyncTask.expression); } dirty = this.$$digestOnce(); if (dirty && !(ttl--)) { this.$clearPhase(); throw "10 digest iterations reached"; } } while (dirty); this.$clearPhase();
  while (this.$$postDigestQueue.length) { this.$$postDigestQueue.shift()(); }};
```

下面是关于如何使用$$postDigest 函数的：

http://jsbin.com/IMEhowO/1/embed?js,console

异常处理

现有对 Scope 的实现已经逐渐接近在 Angular 中实际的样子了，但还有些脆弱，因为我们迄今为止没有花精力在异常处理上。

Angular 的作用域在遇到错误的时候是非常健壮的：当产生异常的时候，不管在监控函数中，在$evalAsync 函数中，还是在$$postDigest 函数中，都不会把 digest 终止掉。我们现在的实现里，在以上任何地方产生异常都会把整个$digest 弄挂。

我们可以很容易修复它，把上面三个调用包在 try...catch 中就好了。

Angular 实际上是把这些异常抛给了它的$exceptionHandler 服务。既然我们现在还没有这东西，先扔到控制台上吧。

$evalAsync 和$$postDigest 的异常处理是在$digest 函数里，在这些场景里，从已列入计划的程序中抛出的异常将被记录成日志，它后面的还是正常运行：

```
Scope.prototype.$digest = function() {  var ttl = 10;  var dirty;  this.$beginPhase("$digest"); do {  while (this.$$asyncQueue.length) {  try {  var asyncTask = this.$$asyncQueue.shift();  this.$eval(asyncTask.expression);  } catch (e) {  (console.error || console.log)(e);  }  }  dirty = this.$$digestOnce();  if (dirty && !(ttl--)) {  this.$clearPhase();  throw "10 digest iterations reached";  }  } while (dirty);  this.$clearPhase();
```

```
  while (this.$$postDigestQueue.length) {  try {  this.$$postDigestQueue.shift()(); } catch (e) {  (console.error || console.log)(e);  }  }};
```

监听器的异常处理放在$$digestOnce 里。

```
Scope.prototype.$$digestOnce = function() {  var self  = this;  var dirty;  _.forEach(this.$$watchers, function(watch) {  try {  var newValue = watch.watchFn(self);  var oldValue = watch.last;  if (!self.$$areEqual(newValue, oldValue, watch.valueEq)) {
```

```
watch.listenerFn(newValue, oldValue, self); dirty = true; } watch.last = (watch.valueEq ? _.cloneDeep(newValue) : newValue); } catch (e) { (console.error || console.log)(e); } }); return dirty;};
```

现在我们的 digest 循环碰到异常的时候健壮多了。

http://jsbin.com/IMEhowO/2/embed?js,console

销毁一个监听器

当注册一个监听器的时候，一般都需要让它一直存在于整个作用域的生命周期，所以很少会要显式把它移除。也有些场景下，需要保持作用域的存在，但要把某个监听器去掉。

Angular 中的$watch 函数是有返回值的，它是个函数，如果执行，就把刚注册的这个监听器销毁。想在我们这个版本里实现这功能，只要返回一个函数在里面把这个监控器从$$watchers 数组去除就可以了：

```
Scope.prototype.$watch = function(watchFn, listenerFn, valueEq) { var self = this; var watcher = { watchFn: watchFn, listenerFn: listenerFn, valueEq: !!valueEq
```

```
}; self.$$watchers.push(watcher); return function() { var index = self.$$watchers.indexOf(watcher); if (index >= 0) { self.$$watchers.splice(index, 1); } };};
```

现在我们就可以把$watch 的这个返回值存起来，以后调用它来移除这个监听器：

http://jsbin.com/IMEhowO/4/embed?js,console

展望未来

我们已经走了很长一段路了，已经有了一个完美可以运行的类似 Angular 这样的脏检测作用域系统的实现了，但是 Angular 的作用域上面还做了更多东西。

或许最重要的是，在 Angular 里，作用域并不是孤立的对象，作用域可以继承于其他作用域，监听器也不仅仅是监听本作用域上的东西，还可以监听这个作用域的父级作用域。这种方法，

概念上很简单，但是对于初学者经常容易造成混淆。所以，本系列的下一篇文章主题就是作用域的继承。

后面我们会讨论 Angular 的事件系统，也是实现在 Scope 上的。

原文 http://www.ituring.com.cn/article/39865

Kb 大小的 JavaScript 的 MVP 框架

摘要: Riot.js 是一个客户端模型-视图-呈现(MVP)框架并且它非常轻量级甚至小于1kb.尽管他的大小令人难以置信，所有它能构建的有如下：一个模板引擎，路由，甚至是库和一个严格的并具有组织的 MVP 模式。

一个可以构建大型网络应用并令人难以置信快和强大但轻量级的客户端框架

Riot.js 是一个客户端模型-视图-呈现(MVP)框架并且它非常轻量级甚至小于1kb.尽管他的大小令人难以置信，所有它能构建的有如下：一个模板引擎，路由，甚至是库和一个严格的并具有组织的 MVP 模式。当模型数据变化时视图也会自动更新。

Riot.js 快而且简单-事实上，是完全不同的规模-而且用它的应用也很快简单。真的。让我详细解释吧。

Riot.js 是目前存在的 JavaScript 模板引擎中速度最快的。它比 Resig 的“微模板”快5倍，比 Underscore 快7倍。

在你应用中的模板影响着渲染时间。模板快可以让你的应用快。在大型应用中影响是很大的，特别是在非 Webkit 浏览器中。上述在火狐中用100K 重复的测试可在 Hogan 花费1300ms 而在 Riot 中耗时30ms。请在不同的浏览器中做做测试可以得到更好的结果。

下面是 Riot 和其它流行的客户端框架文件大小比较： 很明显，Riot 的是最小的，而且框架的大小会影响到以下几个方面：

容易学习。需要看更少的书和指导。这种不同点影响是巨大的：想想看3个 API 和300个 API 的区别。你需要更多的时间来构建应用。这基本上是给大团队用的。错误的决定会导致整个工程失败。

更少专有语法，更多是直接用 JavaScript。用通用的编程技巧代替一些框架自己的规则是很重要的。

更少的问题。更不容易受到攻击和发现弱点。所有的错误都是用户产生的问题点。令人沮丧的是有时候你的应用失败很有可能是因为框架引起的。

易嵌入。在哪里用 Riot.js 都不会让人觉得多余。框架不能比应用本身的代码量大。

更快。根据 Google 的报告，每增加1kb 的 JavaScript，浏览器就多花费1ms 的时间来解析(1)。

更节约。对于 Amazon.com，每增加100ms 的加载 JavaScript 时间会让其税收增加1%(2)。在2012年，Amazon 1%的税收总共$610.9百万。

如果你的工具是其它消费应用而用的，那么其大小可能更加重要

这是一个令人震惊的部分：一个仅有1Kb 的程序库创建一个待办事项的 MVC 应用所需的代码量：

应用所需代码量的图片传达出了一个好框架的能力。一个框架的全部意义在于解决公共问题，这样用户就不必重新发明轮子。写尽量少的代码但能实现更多功能。

代码量的大小取决于多种因素，比如编程风格，所以老实说你的目标不应该是代码量小，而是简单。

简洁是 Riot 真正闪亮的地方。

MVP 设计模式

Riot 使用 Model-View-Presenter (MVP)设计模式来组织代码，这样它能够更模块化、更具可测试性且易于理解。

正如在 MVC（模型-视图-控制器）或 MVVM（模型-视图-视图模型）模式里，其目的是从应用程序的视图中分离逻辑，但 MVP 更简单。让我们把它和 MVC 比较一下：

MVC 模式更复杂。许多箭头围成一个圈。控制器的角色不明确，这种模式可以以许多不同的方式解释。事实上，这是造成有太多该模式下客户端框架的根本原因。

MVP 则相反，没有太多的解释空间，歧义少。每部分的作用是明确的。它适合大大大小小的工程，是单元测试的最佳模式。

让我们看一下 MVP 在 Riot 中是如何工作的。

Model

Riot 的 models 定义了你的应用。 它是你的商业逻辑对外提供的一个深思熟虑后的 API。一个完全隔离的、可在浏览器和服务器端（node.js）运行的可测试单元。下面是一个待办事项应用。

```
function Todo(store) {

    var self = this,
        items = [];

    self.add = function(label) {
        var item = { ... };
        items.push(item);

        // notify others
        self.emit("add", item);
```

```
}

self.remove = function(id) {
    delete items[id];
    self.emit("remove", item);
}

// + other public methods ...

// Enable MVP pattern (this is the secret for everything)
$.observable(self);

// save state
self.on("add remove edit", function() {
    store.put(items);
```

})

}

这个 Model 是一个传统的 JavaScript 对象（POJO），没有框架风格。在 MVC 的术语中，这是一个域模型而不仅仅是数据访问层。

你有权使用 prototype 对象或对象构造器{}。以上只是我个人使用的 JavaScript 风格。当设计一个 model 时，保持思路明确很重要。你需要的最后一件事是用一个框架来聚焦你的应用逻辑。JavaScript 本身就具有巨大的表现力。

observable

观察者是从你应用中分离模型的关键：

$.observable(object);

当一些重要的事件发生后，上面的函数调用给定的对象并通知给其它对象。视图将会重新绘制或者 API 使用者也可以用监听改变事件来予以扩展。

观察者是将你的应用分离为可维护的组件的关键。它是一个典型的设计模式来将模型从视图中分离出来。

"构建大应用的秘籍就是永远不要一下子做成大应用。要将你的应用分割为小的模块"。Justin Meyer,JavaScriptMVC 的作者

"如果将不同组件绑的太紧，可重用性将降低，而且很难在改变一些组件时而不影响其它的组件。"--Rebecca Murphey,jQuery Fundamentals 作者

在客户端框架中一个好的事件库是分离各组件的很重要的特性。这就是 Riot 受关注的原因。

观察者可以对给定对象添加如下方法：

emit(event_name, args...)— 带有可选参数并能触发一个命名事件

on(event_name, fn)— 当一个特定事件发生时调用给定的函数

one(event_name, fn)— 当一个特定事件触发时调用给定函数一次，额外的事件将没有作用

off(event_name)— 停止监听指定的事件

Riot 事件是基于 jQuery 事件的，所以它有很强大的特性，如命名空间，一次可以监听多个事件。

最重要的不同是它没有事件对象，因为它在 DOM 之外就没有相关的事件了。你也能很优雅的发送和接收参数而不用数组来完成。

```
// 发送事件

obj.emit("my-event", "1st argument", "2nd arg", ... argN);




// 接收事件

obj.on("my-event", function(arg1, arg2, ... argN) {




});
```

观察者模式产生于1988年的 Smlltalk 语言，自从那时起它就被用来构建用户接口了。它是一个设计模式，而不是框架。事件和监听器的本质是被分离关注点的优良代码。这也应该是你理念的核心部分。

换句话说：

你不需要用框架来写一个模块级的客户端应用。

视图

视图就是你的应用中可见的那部分。譬如文本，图片，表格，按钮，链接等等，html 代码和 css 样式文件。

Riot 视图尽可能的轻。没有了那些视图中本不该有的条件语句，循环，或者数据绑定，自定义的属性或元素。就是你对所有网页设计师期望的那种视图。

只有"模版"——也就是 html 代码块能在运行的时候插到视图里面。这些模版包含那些可以用 Riot 模版驱动很快用数据替换的变量。

下面是一段 待处理的 MVC 入口：

```
<li id="{id}">
    <div class="view">
        <input class="toggle" type="checkbox">
        <label>{name}</label>
        <button class="destroy"/>
    </div>
    <input class="edit" value="{name}">
</li>
```

这种”不合逻辑的 html “ 没有弱点或可测试的界面。 它通过了 w3c 校验器并且运行更快。

实际逻辑在 presenter 里面.

呈现器

呈现器监听视图上的事件(点击，滚动，按键，返回按钮等等)以及模型(添加，删除或修改的东西)上的事件，当模型更新后视图和模型都会相应的变化。这种"中间人"显示的定义了用户是怎么使用接口的行为。下面是备忘录应用：

```
$(function() {

    / 1. 初始化 /

    // 创建一个模型对象

    var model = new Todo(),
```

```
    // 抓出这个视图 HTML root 实例
    root = $("#todo-list"),


    // 单个备忘录 HTML 模板
    template = $("#todo-tmpl").html(),


/ 2. 监听用户事件 /


// 点击"清除完毕"
$("#clear-completed").click(function() {
    todo.remove("completed");
})
```

```
//点击"切换所有"

$("#toggle-all").click(function() {

    todo.toggle(filter);

})



// ...



/ 3. 监听模型事件 /



// 一个删除入口

todo.on("remove", function(items) {

    $.each(items, function() {

        $(this.id).remove()

    })
```

```
    // 一个编辑入口
    }).on("edit", function(item) {
        var el = $(item.id);
        el.removeClass("editing");
        $("label, .edit", el).text(item.name).val(item.name);
    })


    // ...


})
```

在你的应用中能决定每一个不同呈现器的感觉非常自由。这是基于 UI(边栏，账目，头位置...)上的规则或是基于功能(登陆，添加，创建，删除...)。

呈现器可以用 jQuery 来使用所有的用户接口逻辑。

jQuery

jQuery 绝对是最好的操作 DOM 的工具，而且它也能很好的创建呈现器。jQuery API 设计的很漂亮而且所有的库都非常有用。目前在56%的网站中都有它的身影，92%的网站中用的 JavaScript 库都被人熟知。Riot 将凭借它们引以为豪。

说用 jQuery 可以让代码更糟的错误观点让人失望。没有什么可以超越真理。人们认为代码很乱是因为代码将模型和视图混合了。为了防止此事件发生，通过观察可以简单的分离模型代码和视图代码。你不需要框架的。

用 jQuery 构建 MVP 模式，你所需要做的就是用接口而已。

最糟糕的事情就是因为错误的原因而丢弃 jQuery。它可是高水平的生产高雅和具有可读性代码的 API。其表达式语法简明而且只需要很少的键盘输入。它拥有很大的社区、大量的文档以及它能在所有浏览器上工作。用 jQuery 可以让你感受到自由而不是害怕。

目前构建数据的框架是在 HTML 层次来减少混乱代码的。突然间 onclick 属性隐藏在后面了(我惊讶的看着你)。编程书这样告诉我们，要将事物分离开来，在 web 开发中的意思如下：

JavaScript 要在 CSS 之外

CSS 要在 JavaScript 之外

JavaScript 要在 HTML 之外

HTML 要在 JavaScript 之外

Riot 是简洁的方法，它不会让你在 HTML 视图中参杂任何逻辑。

URL 的返回按钮

另一个让人困惑的来源是返回按钮。按照一般的约定,你需要一个叫“路由器”的工具。

其实这是不对的。点击返回按钮或者改变 URL 是一个你可以监听的全局窗口事件。它应该和按下一个键盘按键事件一样。目前的路由器是一个为很基础功能而产生的巨大解决方案。没有必要将工具和处理 URL 事件相分离 ,这就像没有必要将工具和处理按键事件分离一样。

上面的路由器产生了混淆是因为它不符合典型的 MVC 模式。在之前的网络时代，传统的 MVC 应用没有 URL。

Riot 只有一个$.route 函数来处理应用的状态和返回按钮

```
// 点击导航元素
$("#filters a").click(function(e) {


// 跳过链接默认行为
e.preventDefault();
//改变 URL 哈希部分
```

```
    $.route($(this).attr("href"));

})

 //监听 URL 改变事件

$.route(function(hash) {

    /*

        待办事件列表处理：

        1. 清除列表并添加新项

        2. 高亮导航上激活的元素

        3. 更新视图计数器

    */

})
```

你只需要一个函数。我把它称作人们所期待的“路由”。它的作用如“url”或“go”。在 url 参数的基础上，你可以有一个或着多个 URL 监听器来完成其功能。

如果你的应用严重依靠不同哈希参数，你可能需要一个全面的路由器，但在大多数应用中是不需要的。

这就是 Riot！

目前客户端框架都存在以下错误的假设：

存在一大块常见的问题

MVC(或 MVP)需要一个框架

jQuery 让代码更加混乱

所有这些陈述都是相当的错误。

目前的框架都会劝说大家相信错误的信条，即在没有易懂的、科学的分析下可以创建好的网站和非常精致的营销。为了解决假设的问题而会产生新的问题。很不幸，大量的通信在处理不相干的问题点。

Riot 就是反对现状！

目前的应用可以更加的快、简单和小巧。它们可以让更少的人维护，需要更少的知识和出现更少的错误。

模型应该是普通的 JavaScript，视图应该是普通的 HTML。它们两都应该独立的发展而互不影响。框架没有必要负担开发和命令事情如何发展。

没有“Backbone 的方式”、“Angular 方式”或是“Ember 的方式”。框架一直都是经典的编程技巧。

Riot 是对平凡的 JavaScript 和 jQuery 的一种宣言。接下来呢？

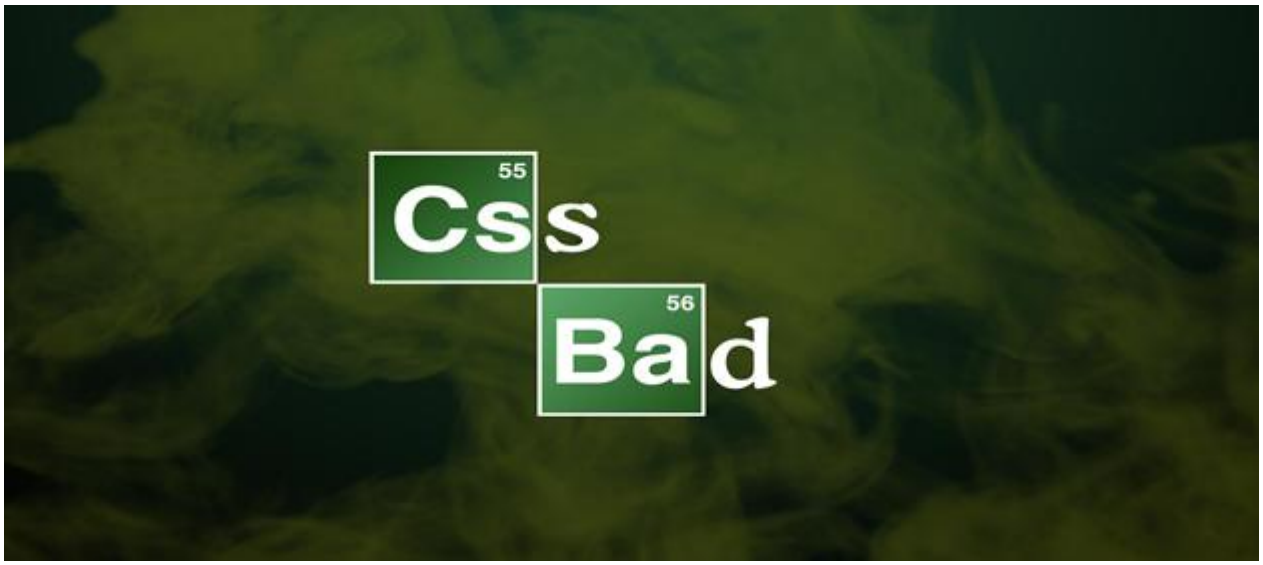
我 之前的一个博客主题是有关 Riot.js 和无框架化的 JavaScript。我想要用无框架结构来控制应用。事实上，只需要3个函数即可。

我的下一篇博客主题则更深层次的讨论了用 Riot 构建现代化的 JavaScript 应用。你可以学习到 Riot 怎么处理测试，持久化，应用辅助程序，资源分享，以及文件操作。

最后，官方的 Riot.js 即将出发行版，并且它有为浏览器和 nod.js 发布了文档和 Moot 论坛。关注我们的 Facebook 页面和 Twitter 公告

原文 http://www.html5cn.org/portal.php?mod=view&aid=5736

网页重构应该避免的10大 CSS 糟糕



今天在回广州的火车上看到一篇"10 CSS mistakes every web designer must avoid"，想着还不错，就翻译给大家。

对于网页重构来说，CSS 禅意花园 是网页布局从 table 表格转到了 html +css 的标志 。这些年来，随着我们的网站越来越复杂：html5，css3，新的技术、新的属性，越来越多的开发者开始思考和尝试提高他们的 CSS 技能。那么我们从哪里着手呢？对于网页重构工作

来说，我们应该养成什么样的开发习惯？一个糟糕的 css 用法是怎样的？我们应该怎么处理这些糟糕的 css。今天这篇文章，我将谈一谈10个我们应该避免的 css 糟糕用法，当然，我们也会分享怎么才是正确的用法。

为了方便大家理解，我将这10个糟糕用法归为三大类：权重、工作流、自以为是。

权重！权重！！权重！！！

正所谓马太效应，如果你写了很烂的 css，这段烂代码的不好之处是他会导致更多和更烂的代码。如果你需要解决一个 css bug，发现唯一的解决方法是：使用更多层级的选择器、 id 选择器；甚至更糟糕：使用内联样式(inline-style)，直到使用最后的大杀器!important。以上提到的所有用法归根到期他们犯了「过多权重」的错误。

CSS 中的权重取决于你如何使用具体的 css 规则。

```
#layout #header #title .logo a { display: block; }
```

看看上面这串 CSS 选择器，你可能觉得这是一段符合语义化的「好」选择器：简单明了。如果你依照习惯从左往右读，你可能是这么理解：「先找到主要布局区域，再找头部区域，再找到 title 标题区域，再找到 logo 标志区域，再找到这里面的所有 a 锚点」，对于 css 来说，这是一个很具体的精确定位，但是，实际上我们很少有机会需要这么精确的定位。过大的 css 权重会造成你的样式表更加难以维护（你考虑过接你班的同事的感受木有！），也更加难以阅读和理解（这么多层声明、一长串你闹哪样啊）。

01.滥用 id

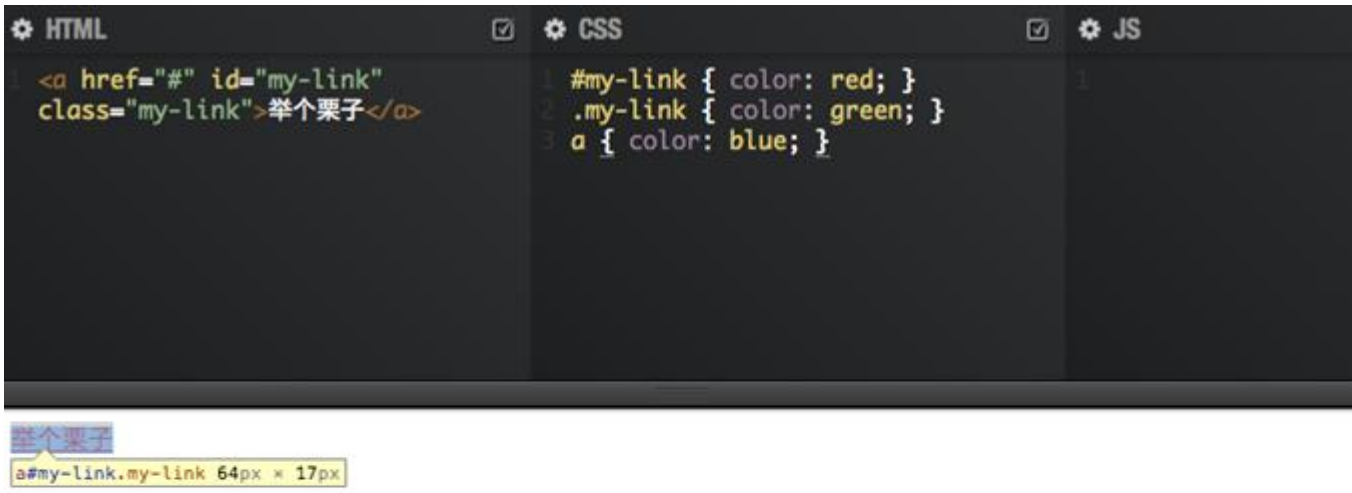
CSS 权重的高低取决于你使用什么样的选择器：id,class 类,tags 标签。举个栗子：

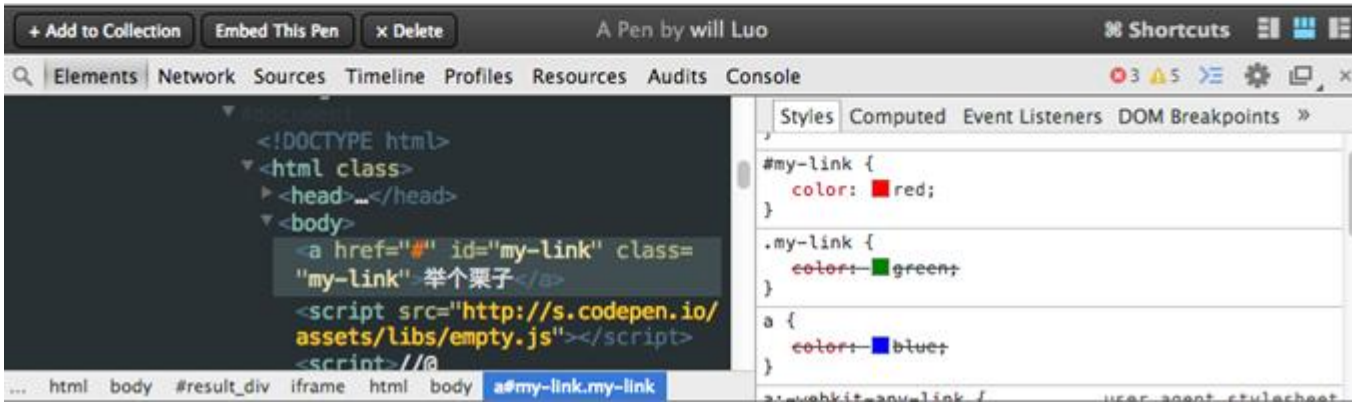
#my-link{color: red}
.my-link{color: yellow}

a{ colour: blue}

来做这么一个小测试，有这么一个超链接如下，如果我们没有其定义其他样式，浏览器渲染出来的最终结果会是什么颜色？

举个栗子





最终的颜色会是红色，因为 id 属性是三者之中权重最高的（id 在网页中只能使用一次，他得权重值为100），所以红色葫芦娃成功击败了其他娃娃，举起了栗子。

根据 css 权重，权重次于 id 的是 class 类，最后才是 tag 标签，正如你如上图审查器中所看到的。

当然啦，像上面我们举的栗子这种「同时使用 id/calss/tag 」的情况在实际应用中还是很少见的，在日常开发中，我们更多的情况是会遇到如下情况：

```
#header a { border:2px dashed #000 }
```

假设这是我们的一个项目，现在我们决定要把一个在 header 的 link 设置成无边框，随手一写，我们添加了：

```
.special-link { border:none }
```

然后再在 html 中添加了一个 special-link 的 class 类，这下解决我们的问题了吗？ 答案是：没有！ 由于 id 的权重如此之高，我们需要更高权重的声明才能实现我们的需求。下面这样写才是正确的：

```
#header .special-link { border: none }
```



再举个栗子 特殊的栗子



假如说这种情景在我们的 code 过程中不断地出现：设置 header 区域另外一个特殊的超链接 link 为某特殊的样式，你该怎么处理呢？ id 的高权重特性意味着滥用 id 绝对是一个很糟的做法！

那我们如何解决这个问题呢？用 class 类来代替 id 吧。

02.大串的 css 选择器（多层级）

```
#header #title .left-side img.logo { opacity: 0.5 }
```

上面这个栗子不仅仅乱用 id，他还很长，正如同乱用 id 导致的高权重麻烦，如果你使用一大串选择器（超过三个层级）你同样会造成过高权重，导致你陷入到高权重和更高权重的汪洋大海之中。

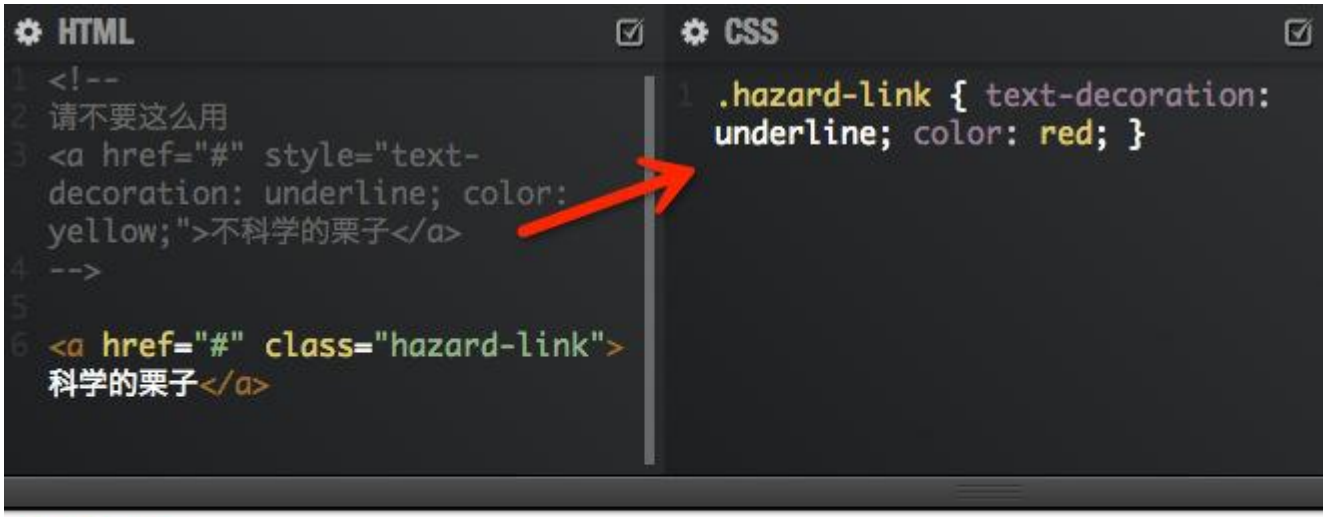
那针对这个问题的解决办法有什么呢——精简！如同我们现在这个栗子，如果我们用一个 `.logo` class 类就能解决我们的需求，那我们就没必要写这么以大串了。一般情况下，我们应该控制选择器在2个，最多3个。

03.Inline styles

内联样式是 css 权重罪恶的源泉，同时也从根本上摧毁了我们使用 css 的初心（结构样式分离）。当我们的工程师好不容易走出 tables 的魔窟开怀拥抱外部样式表，我们早就应该知道不要把样式同结构混杂在一起。

根据 css 权重级的特性，内联样式只能被`!important` 所覆盖。一般来说，这就意味着，如果某猿使用了`!important`，他们更多是被逼的没办法才这么用(对应英文 reactive)，而不是想这么用(preoactive)。的确`!important` 在 css 样式表中用起来十分方便，但我们最好是聪明地、小心翼翼地碰她、用他，而不是把他当做救命稻草（救命稻草用多了，迟早也会从救命稻草变成压死骆驼的稻草）。

下面举例怎么解决内联样式的问题，就两步 1.复制删除 2.粘贴 。剔除内联样式，转移到样式表之中吧。



04.从上至下式的粗放命名

说完 css 权重，接下来我们来谈谈其他 CSS 的糟糕用法。假设我们开始了一个新项目，设计师丢给我们一份 psd，我们开始在 html 中写基本的框架。

根据结构从下至上式的命名方式模糊化了 html 结构，工程师常常根据上下结构来命名 id 和 class，而不是具体的设计元素，例如#header,content，这常常会导致长选择器（举个例子如. menu ul li a { }），这样的后果是我们的代码变得难以调试和维护。怎么解决这个问题呢？我们应该尝试从网页中分离出设计元素，这同样可以减少我们代码中的冗余。

05.冗余/重复

冗余意味着你写 css 的过程中不断重复某些代码。在使用编程语言的时候， 我们很好理解重复（造轮子）意味着浪费时间，我们在 code 中应该遵循 DRY 原则(Don't repeat yourself)。什么情况下我们会重复造轮子呢？举个栗子：

```
.some-title { font-weight: bold; }
```

```
.some-other-title { font-weight: bold; color: red }
```

实际上，我们可以有个更好的解决办法

```
.some-title, .some-other-title { font-weight: bold; }
```

```
.some-other-title { color: red; }
```

比如说我们要添加一个不同颜色的标题，我们可以使用一个常用的命名，或者添加一个具体的 class 类。

```
<h3 class="some-title pop">我的标题/h3>
```

这个有点**面向对象的 CSS** 的思想在里面，使用 Sass 中的 @extend 特性可以很好地解决我们这个问题。

在 Sass 之中，你可以使用@extend 继承选择器，被继承的选择器的样式也被继承。这个特性使得我们可以很方便管理不同的样式，举个栗子：

```
.some-title { font-weight: bold; }
```

```
.some-other-title { @extend .some-title; color: red; }
```

当 CSS 预处理编译器将.scss 转换成.css 文件时，我们最终输出的样式是：

```
.some-title, .some-other-title { font-weight: bold; }
```

```
.some-other-title { color: red; }
```

最终输出的是一模一样的效果，解决重复和冗余的问题，要求我们在写 css 的时候心中对 html 层级结构要有个大致的规划，思考不同的设计元素之间的层级和关系，我们规划得越清晰，最终输出的 css 也越精简。

06.精简你的单位

如果你的样式表中混杂着 px,em,rem 等等单位，是时候改变了，业内著名的 web 开发者 Rachel Nabors 呼吁大家统一使用 em 为字体大小单位，他曾说「我看其他人的样式表第一件事就是看字体样式，然后把所有的 font-size 的单位换成 em」，这几年用户使用的终端越来越多样（不同的终端、不同的浏览器使用的默认字体大小存在差异），使用绝对字体大小 px 变得越来越不可控，使用 em 等相对大小的字体则避免了这个问题。

如果你想在转成 em 对这些单位有个深入点的了解，推荐你阅读《CSS 文字大小单位 PX、EM、PT》。

当然，如果你就是不喜欢 em 和他们嵌套特性，那么你可以更进一步了解 CSS3 中推出的新单位 rem。

不同单位在 web 设计之中的战争还在继续(可参考文章 CSS Font-Size: em vs. px vs. pt vs. percent ），学习一点相关知识对我们提高代码可维护性至关重要。

下面我们进入到这篇博文的第三也是最后一个部分「自以为是」，这些坏习惯包括：增加不必要的东西，错误，无意义的 css。

07.向下兼容和无效的规则

在开发的过程中，如果你不需要使用 css3 之类的高大上代码就能实现效果 ,那再好不过了。可是，作为工程师的你在 Chrome 最新版本上面看到的效果，并不意味着你的用户能在他

们的浏览器上看到同样的效果（考虑过 IE 的感受没有），一个十分糟糕的坏习惯就是完全忽略向下兼容性。

如果你在项目中使用 rgba()，你是否测试过这个属性的兼容性？如果没有，那你最好祈祷没有 IE8的用户会访问你的网站。你是否写全了针对不同浏览器的不同的规则（-webkit,-ms -moz 等等）？解决这个问题，可以使用 CSS LInt 检测下你的 css 代码，CSS Lint 的检测规则包括错误的和不合理的地方。

08.(没有意义)不起作用的样式

Harry Roberts 的 Code smells in CSS 是关于 css 糟糕用法的经典文章。他举了几个可有可无的不起作用样式的栗子：

```
h2{
font-size:2em;
margin-bottom:0.5em;
padding-bottom:0.5em;
border-bottom:1px solid #ccc;
}
.no-border{
padding-bottom:0;
border-bottom:none;
}
```

Roberts 推荐的重构精简方法是删掉属性为0和 none 的属性值。

```
h2{
font-size:2em;
margin-bottom:0.5em;
}
.headline{
padding-bottom:0.5em;
border-bottom:1px solid #ccc;
```

```
}
```

如果你像重构之前的那样写法，h2是一个我们在 web 设计中会不断重复用到的元素，这个本质上「你写了更多的代码却没有实现了更少的样式效果」，如果我们接下来又要设置一个h2的属性为其他样式，那代码会得很乱。



Harry Roberts 是 inuit.css 框架的作者。

09.巧而不巧:用 Hack 不意味着你是个好 Hacker

负的 margin 边距，!important 等等，上面的这些就是我们所说的 hack 用法（此 hack 非针对 IE 兼容的 hack，也可以理解成 cheater 作弊用法），如果其他人问我们为什么要这么写？我们可能只需要回答「管他呢，反正能实现效果」。

当我们为自己使用这些充满「弊」端的方法而略不放心的时候，一个解决办法就是把我们的这些 hack 放到一个特定的样式表文件 hack.css 之中。

任何时候你意识到你写了一个 css 的 hack 用法的时候，你直接把这些代码放到这个 hack.css 之中（或者样式表的特定区域：通过注释跟其他样式区分开），这个专属区域是个好解决方案因为他最终会在用户端隐藏。

当我们养成了这个习惯，我们可以十分清晰地知道我们写了哪些 hack ，同样可以方便我们了解我们使用这些 hack 的情景，让我们可以知道如何避免这些 hack 。我们有许多写 hack 用法的理由 ,但是如果我们不记录我们为什么 hack ,我们将不会从我们这些 hack 中学到我们为什么要错误地这么用。

10.糟糕的文档和注释

昨天在微博上看到一个段子「程序员最讨厌的四件事：写注释、写文档、别人不写注释、别人不写文档 …」。

写文档和注释绝对不是一个有意思的事情，但却是一个最重要的事情（尤其是涉及到项目后续可维护性时），文档可以有效地让其他人知道你的代码是干什么的，同时其他人理解你的 css。对于大部分语言（ html,css,php,JavaScript ） ，开发者可以直接在代码文件中写上注释（ 文档 ）。

Nabors 曾说「我曾 YY：如果我今天写完一个项目的 css 但是明儿我却挂掉了，有一个人~~幸运地~~接手我这个项目，那他看得懂我的这些代码是什么意思吗？」。尽量地让我们的样式表中的结构清晰，如果不能让人立马知道你的选择器指的是哪里的样式，可以尝试添加注释（什么！你还说注释增加 css 文件大小！难道你不知道压缩工具么！）。

第一步是在必须的地方做好注释，第二步是使用工具把 css 文件中的这些注释转换成一个合适的文档。推荐可以使用 css doc 和 KSS。

css_doc 跟 JavaDoc 类似，他的转换原理基于 CSSDOC. KSS（Knyle Style Sheets），对于前端和设计师来说都十分有益。

总结

在这篇文章中，我们介绍了一些常见的 css 糟糕用法，指出了为什么我们应该避免这些用法和应该使用什么正确的用法。还是这句话"Doin something is always better than nothing"，行动起来总比什么都不做要好点，未雨绸缪有备无患。

今天你看完这篇文章，知道了10个糟糕的 CSS 用法，这不意味着你明儿去上班就把你过去所有的代码都重构一遍。从点滴开始，一步一步来才是我们开发工作的正确做法。在接下来的工作中，注意这些细节，避免这些「小」错误，让我们的代码更漂亮点。终有一天，我们会发现"Code is poetry"。

译者言

这是第二次翻译技术相关文章，不得不说，还是低估了翻译的难度。要把技术类的文章翻译得语言流畅、通俗易懂，还是挺有挑战性的。译文中加入了自己的理解，配图也是自己通过 codepen 重新截图演示，这篇文章的译稿我放在了 google docs 上，如果对于某些翻译你有更好的用法，可以直接建议修改。

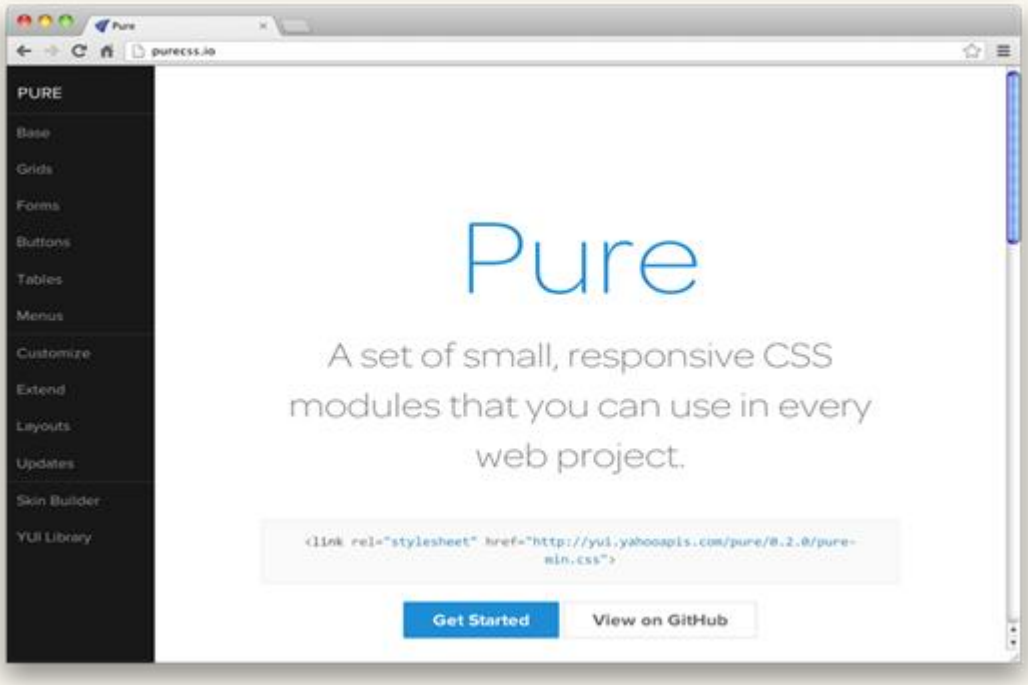
原文 http://jianshu.io/p/jG8GEu

值得拥有！精心推荐几款超实用的 CSS 开发工具

当你开发一个网站或 Web 应用程序的时候，有合适的工具，绝对可以帮助您节省大量的时间。在这篇文章中，我为大家收集了超有用的 CSS 开发工具。

对于 Web 开发人员来说，找到有用的 CSS 开发工具，就像找到一个魔灯，可以让他那些艰难的任务变得有趣。CSS 开发工具帮助开发人员创造更加时尚，更有创意的网站。

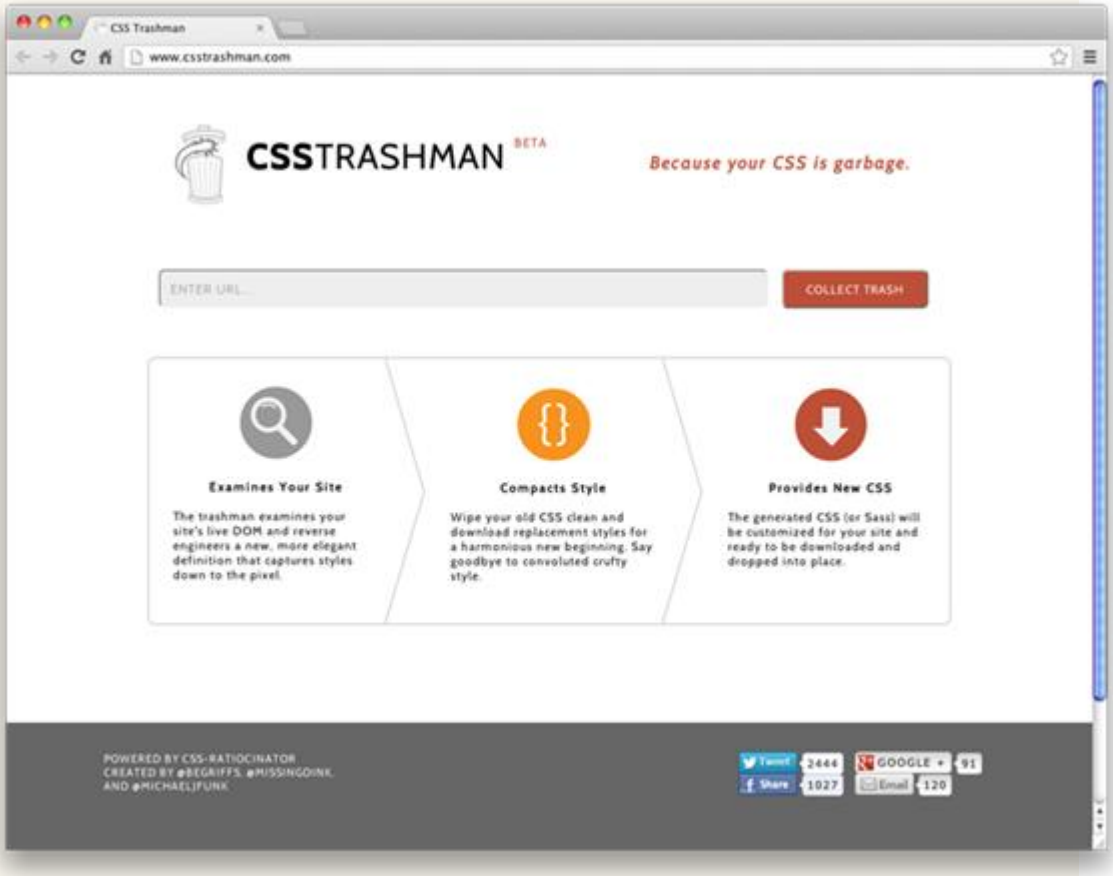
Pure CSS



Pure CSS 来自雅虎，是一组 CSS 模块，您可以作为基础在每一个 Web 项目中使用。它使 CSS 开发更容易，整个框架看起来很干净，专业。如果你还没有用过的话，尝试一下！

更多信息: http://purecss.io/

CSS Trashman



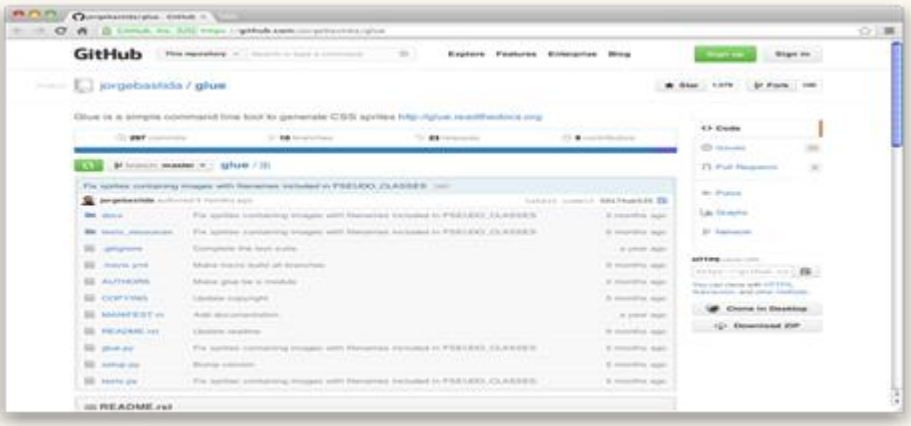
在大的项目中，保持 CSS 文件的简洁和紧凑是非常空难的。不过不要担心，CSS Trashman 可以帮助你。这个在线工具检查您的网站，压缩样式，让你下载一个超级干净的版本的网站 CSS 文件，这样你就可以安全地删除它们。更多信息: http://www.csstrashman.com/

Bear CSS



Bear CSS 是一个美丽的网站，也是一个超级有用的工具。只需创建一个 HTML 文档，并把它上传。Bear CSS 会自动在 HTML 元素的基础上生成一个可下载的 CSS 文件。更多信息: http://bearcss.com/

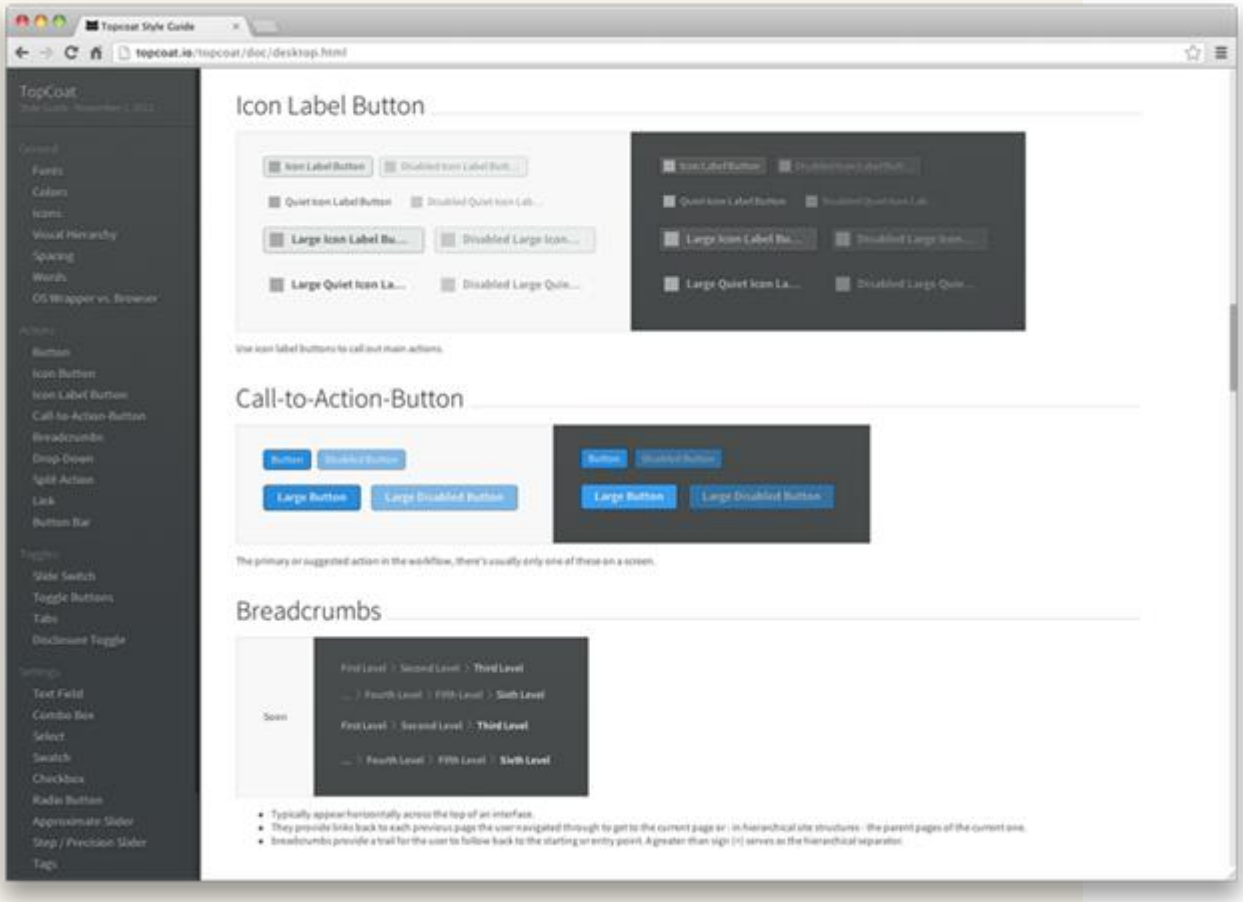
Glue



CSS 精灵技术是一种减少 HTTP 图片请求的很好的方式。Glue 是一种简单的命令行工具，转变一组图像到一个文件中，因此您可以轻松地在您的网站上使用 CSS 精灵技术，超有用！

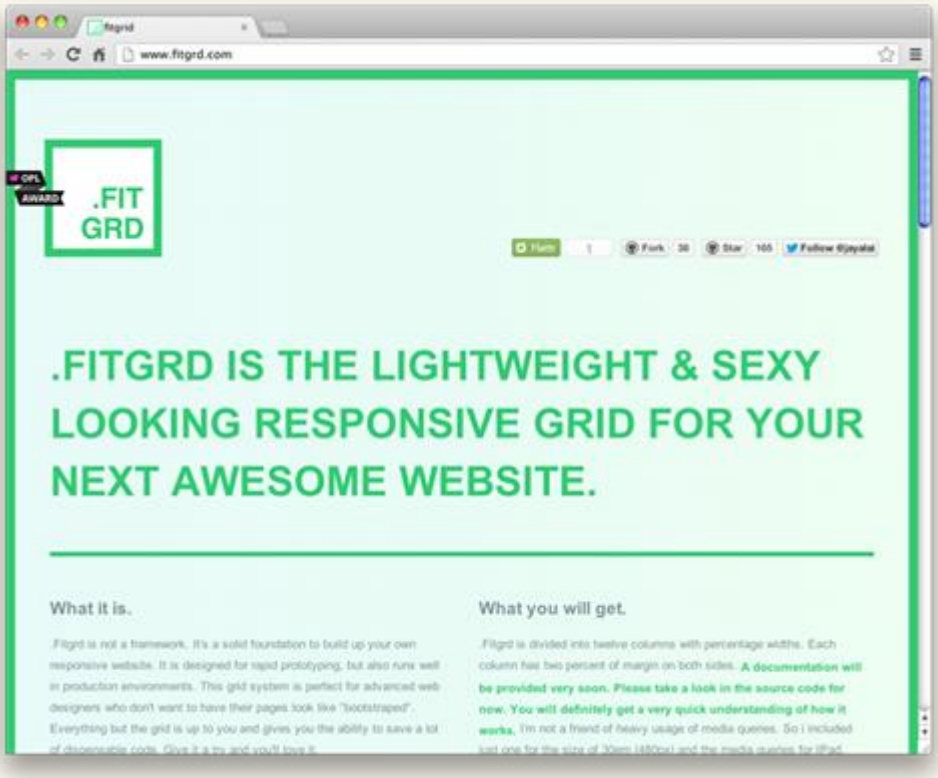
更多信息: https://github.com/jorgebastida/glue

Topcoat



Topcoat 是一个 CSS 集合，用于简洁和快速的 Web 应用程序开发。它是一个开源库，不需要任何 JavaScript 内置的用户界面元素。更多信息: http://topcoat.io/

.Fitgrd



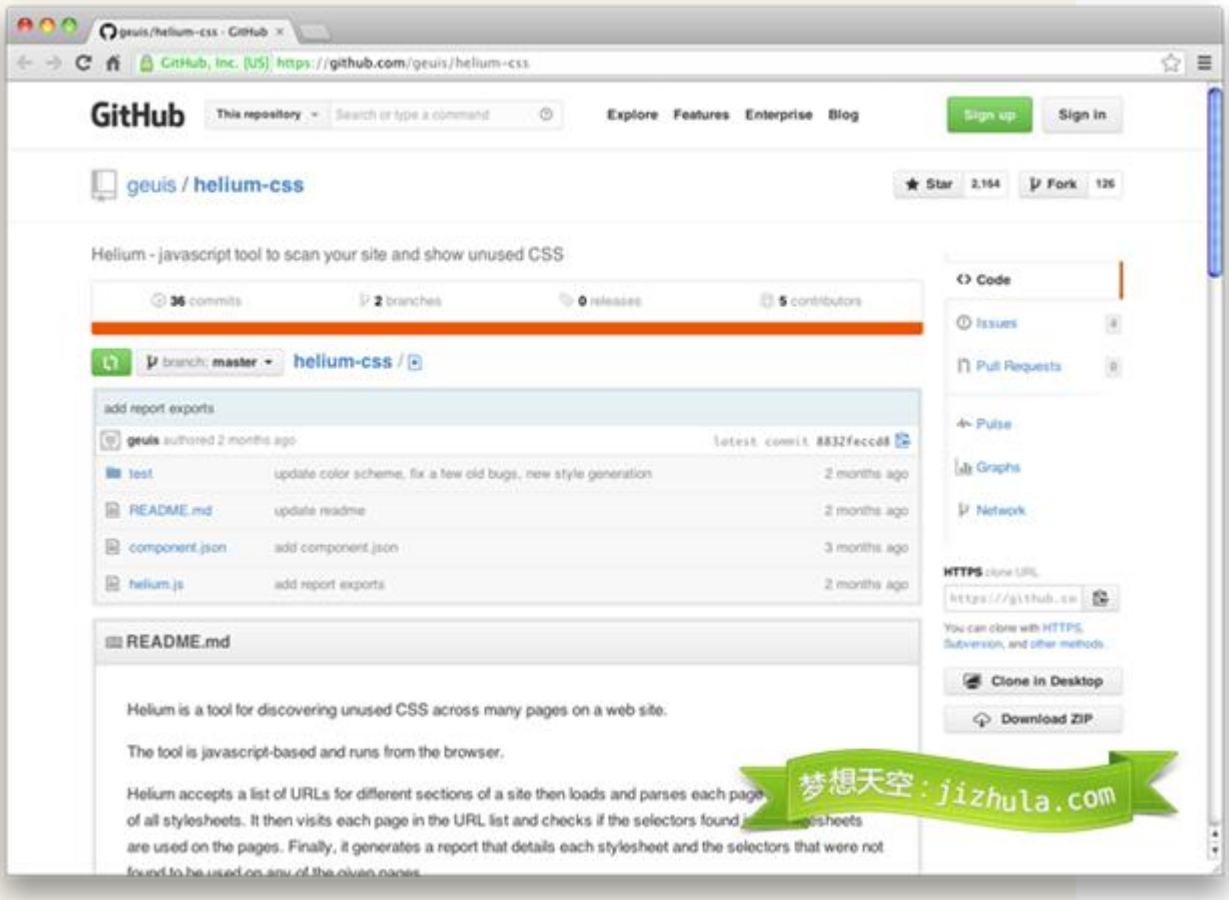
.Fitgrd 是一个轻量的响应式网格系统，是建立你自己的响应式网站的坚实的基础。更多信息: http://www.fitgrd.com/

Normalize.css



Normalize.css 是用于 HTML5 的 CSS 重置样式，它使浏览器呈现的所有元素更一致和符合现代标准。它被用于许多流行的网站，如 Twitter Bootstrap, Soundcloud, TweetDeck 等等。更多信息: http://necolas.github.io/normalize.css/

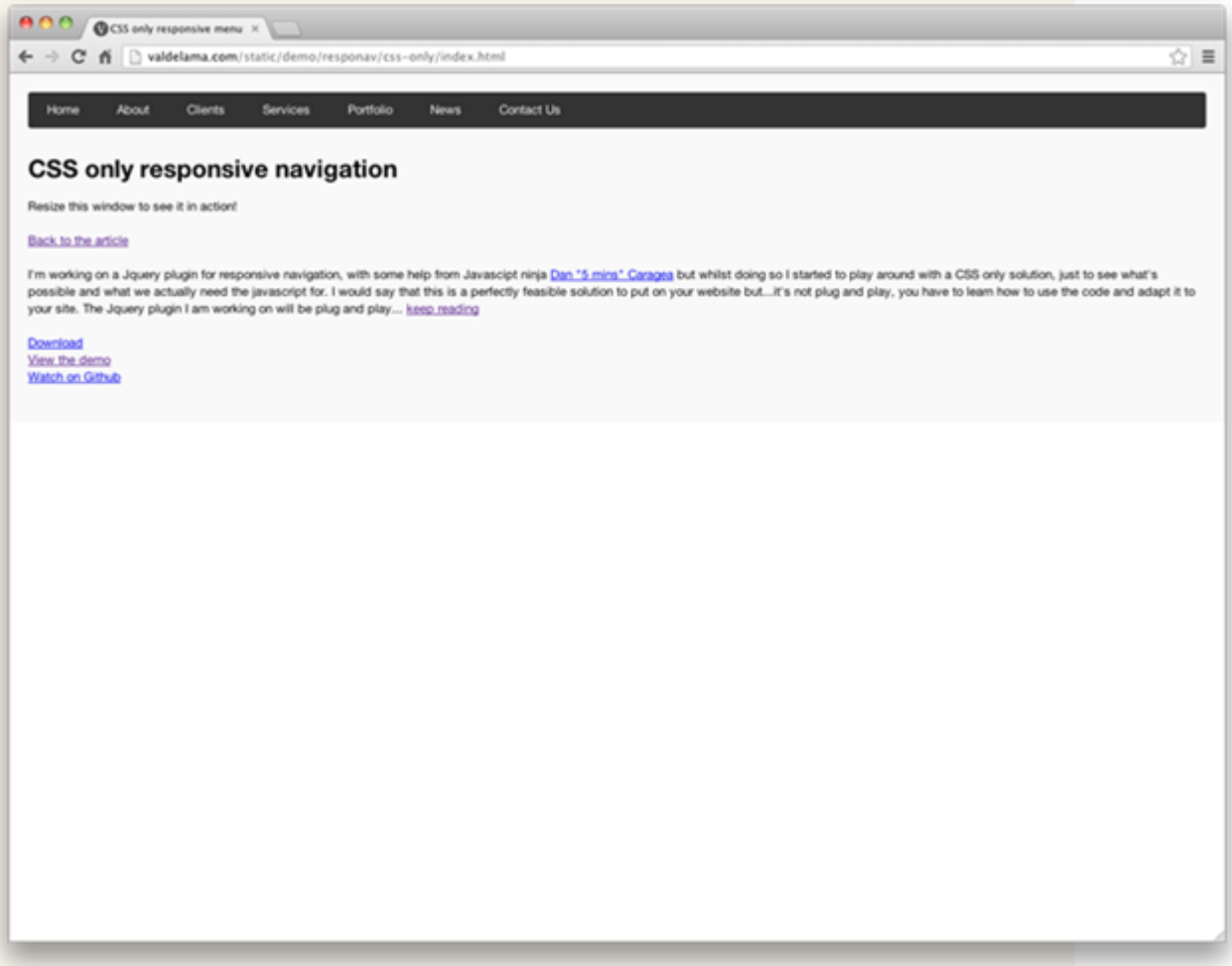
Helium CSS



在大的样式文件中，很难找到未使用的样式。Helium 是一种基于 JavaScript 的工具，运行于浏览器，可以从你的 CSS 文件检测未使用的样式。更多信息:

https://github.com/geuis/helium-css

CSS only responsive navigation



响应式的布局，现在在网络上很受欢迎，在我看来，这绝对是一件好事。这里是一个可下载

和可重复使用的响应式 CSS 导航菜单。更多信息:

http://valdelama.com/css-responsive-navigation

原文 http://www.cnblogs.com/lhb25/p/super-useful-tools-for-css-coding.html

CoffeeScript 尝鲜

CoffeeScript, JavaScript, node.js

很久没有更新博客了，因为没有什么感觉。

最近要从头开始实现聊天室应用，打算用 node.js。这几天都在调研相关的技术，碰到很多 node.js 的项目都是用 CoffeeScript 写的。出于好奇，便跑到 CoffeeScript 的官方看个究竟。

看了之后发现 CoffeeScript 写起来挺简捷的。语句不用分号结束，函数调用不用写括号。代码块不需要大括号，而是像 Python 那样用缩进表达。google 了一些资料，很多人都说 CoffeeScript 是 Python 和 Ruby 的混合体。但我看到的第一感觉像 Erlang，因为函数调用都是用`->`。把自己尝试写的一段 CoffeeScript 发给一个用 Erlang 做页游的同学看，他的第一感觉也是像 Erlang。

看着感觉很爽，于是就试着边看文档边写一段试试。就随便到 node.js 文档里找了一段示例代码 http://nodejs.org/api/http.html#http_http_request_options_callback 进行改写，也便于对照是否有误。

```
options =
hostname: 'www.google.com'
port: 80
path: '/upload'
method: 'POST'

req = http.request options, (res)->
console.log 'STATUS: ' + res.statusCode
console.log 'HEADER: ' + JSON.stringify res.headers
res.setEncoding 'utf8'
res.on 'data', (chunk)->
console.log 'BODY: ' + chunk

req.on 'error', (e)->
console.log 'problem with request: ' + e.message

`// write data to request body`

req.write 'data\n'
req.write 'data\n'
req.end()
```

在 CoffeeScript 官网的即时翻译页面写了，发现的确快了不少，而且嵌套的回调代码也没有之前那么碍眼。大部分特性都比较好，就是不支持单行注释。用#注释只能在 CoffeeScript 看到，翻译成 JavaCcript 之后就没有了。如果想保留注释到 JavaScript，只能用``来括起来，以原样输出。但还是有点美中不足，就是注释最后会多出一个分号。

写好的 CoffeeScript 保存成 test.coffeescript。用翻译器可以翻译成 JavaScript：

```
coffee -c -b ./
```

编译出来的 JavaScript 代码如下：

```
var options, req;

options = {
hostname: 'www.google.com',
port: 80,
path: '/upload',
method: 'POST'
};

req = http.request(options, function(res) {
console.log('STATUS: ' + res.statusCode);
console.log('HEADER: ' + JSON.stringify(res.headers));
res.setEncoding('utf8');
return res.on('data', function(chunk) {
return console.log('BODY: ' + chunk);
});
});

req.on('error', function(e) {
return console.log('problem with request: ' + e.message);
});

// write data to request body;

req.write('data\n');

req.write('data\n');

req.end();
```

从编写的过程中发现，CoffeeScript 带来了比 JavaScript 更多的特性。不过与 JavaScript 相比，CoffeeScript 提供了一套独立的语法，即使没用过 JavaScript 也可以直接上手。代码逻辑的表达及书写速度的体验都比 JavaScript 好。

原文 http://gutspot.com/2013/11/17/coffeescript 尝鲜/

一个关于 Zend O+的小分享

这个是我上周末在”阿里 PHP 技术沙龙”临时分享的一个主题的 PPT，主要是介绍一下 Zend Optimizer Plus(简称 O+).

O+是由 Zend 公司开发的一个 PHP 性能提升工具，在 PHP5.5开始，已经随着 PHP 的源代码一起发布了，并且也改名为:Opcache.

不同于 APC，O+除了是 Opcodes Cache 以外，还做了很多的 Opcodes 优化，这个 PPT 就是主要列举了一下主要的优化们.

也不同于 eacc, O+做的优化更多一些.

这个特性，就使得 O+要比 APC 快不少，从 Benchmark 来看(Zend/bench.php)，O+能节省5%的 CPU IR，能快7%以上.

另外就是，随着 Opcache 进入 PHP 源代码树，APC 我们也就不在做更新了，主要的精力都迁移到了 O+的进一步开发上.

所以，建议大家都切换 O+吧(支持5.2 到最新的5.5). 基本上只需要改一个配置，你就能得到性能提升，何乐而不为呢？是吧.

一个 Opcache 的推荐配置:

```
zend_extension=opcache.so
opcache.enable_cli=1
opcache.memory_consumption=128 //共享内存大小，这个根据你们的需求可调
opcache.interned_strings_buffer=8 //interned string 的内存大小，也可调
opcache.max_accelerated_files=4000 //最大缓存的文件数目
opcache.revalidate_freq=60 //60s 检查一次文件更新
opcache.fast_shutdown=1 //打开快速关闭，打开这个在 PHP Request Shutdown 的时候
                                                    // 会收内存的速度会提高
opcache.save_comments=0 //不保存文件/函数的注释
```

对于5.5以前的 PHP，可以在这里下载到最新版的 O+, ZendOpcache

Zend Optimizer Plus(O+) from Xinchen Hui

原文 http://www.laruence.com/2013/11/11/2928.html

Django 源码小剖: Django ORM 查询管理器

ORM 查询管理器

对于 ORM 定义：对象关系映射, Object Relational Mapping, ORM，是一种程序设计技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换。从效果上说，它其实是创建了一个可在编程语言里使用的“虚拟对象数据库”。ORM 能大大简化并抽象数据库的操作.

```
# views.py
def index(request):
    book_set= Book.objects.filter(id=1)
    或者
    book_set= Book.objects.all()
    ......
```

假设 django 的一个工程中包含一个名为 Book 的模块(model)，在 views.py 的函数中可能会写出查询语句:

ORM 的作用就是这样，并不需要写更复杂的 SQL 语句，所有的的事情都被 ORM 代劳了.

上面中, Book 实际上是一个 Model 实例，但先是从 Book.objects 开始说起.

Book.objects 实际上是一个 Manager 类实例，每个 Model 都会有一个，用户的查询操作几乎是从这里开始的. 万万可以将 Model 实例理解为关系表中的一个表项数据，而 Manager 实例可以理解数据库查询的入口.

这个函数确保每一个 model 都有一个管理器 objects

```
def ensure_default_manager(sender, **kwargs):
    ......

    if not getattr(cls,'_default_manager', None):
        # Create the default manager, if needed.
        try:
            cls._meta.get_field('objects')
            raise ValueError("Model %s must specify a custom Manager, because it has a field named 'objects'"
cls.__name__)

        except FieldDoesNotExist:
            pass

        """
        关键的一步，将一个 Manager 实例挂钩到 cls.objects，将 model.add_to_class() 方法罗列如下;
        def add_to_class(cls, name, value):
            if hasattr(value,'contribute_to_class'):
                value.contribute_to_class(cls, name)
            else:
                setattr(cls, name, value)


        关键是 Manager 有 contribute_to_class() 方法，由此看来，model.objects 并不是一个 Manager 实例，实际_
```

他是一个 ManagerDescriptor 实例.

```
        """
        cls.add_to_class('objects', Manager())
        cls._base_manager = cls.objects

    elif not getattr(cls,'_base_manager', None):

        default_mgr = cls._default_manager.__class__

        if (default_mgr is Manager or
                getattr(default_mgr,"use_for_related_fields", False)):
            cls._base_manager = cls._default_manager

        else:
            # Default manager isn't a plain Manager class, or a suitable
            # replacement, so we walk up the base class hierarchy until we hit
            # something appropriate.
```

```
for base_classin default_mgr.mro()[1:]:
    if (base_class is Manager or
            getattr(base_class,"use_for_related_fields", False)):
        cls.add_to_class('_base_manager', base_class())
        return
```

实际上，无论如何都在 Model 类的源码中找到任何 objects 属性的字眼，因此它肯定是在某个时间点上增加的. 可以在 django.db.models.manager.py 中找到下面的函数:

由此可以发现, Model.objects 在这个时候被添加了. 因此用户可以在代码中使用 Book.objects. 至于这个函数在何时被调用, 待后面会详述 django 内部的信号机制. 暂且你可以将其理解为在 django 服务器启动的时候, 这些会被自动调用就好了.

Manager 实现

Manager 保护技法

如果可以在 book_set = Book.objects.filter(id=1) 这一句上设置断点, 并 step into 的时候, 发现 Book.objects 实际上的实际上不是一个 Manager 实例, 而是一个 ManagerDescriptor 实例, 这是 django 特意为 Manager 做的一层包装. 为什么要这么做 ?

django 规定，只有 Model 类可以使用 objects，Model 类实例不可以. 请注意区分类和类实例之间的区别.

我认为这样做是有道理的. Book.objects.filter(id=1) 返回的是 QuerySet 对象, 而 QuerySet 对象可以看成是 Model 实例的集合, 也就是 book_set 是 Model 实例的集合. 假使 「Model 类的实例可以使用 objects 属性」, 即「从一本书中查询书」这在语意上不通过. 只能是「从书的集合(Book)中查询书」.

```
class ManagerDescriptor(object):
    """
    很经典的掩饰，为 Manager 特殊设定 Descriptor，从而，只能让类访问，而不能让类的实例来访问. 具体是靠
instance, type=None) 方法来实现来的: 第二个参数 instance，当 class.attr 的时候，instance 为 None; 当 ob
instance 为 obj.
    """
    # This class ensures managers aren't accessible via model instances.
    # For example, Poll.objects works, but poll_obj.objects raises AttributeError.
```

```
def __init__(self, manager):
    self.manager= manager

def __get__(self, instance,type=None):
    if instance != None:
        raise AttributeError("Manager isn't accessible via %s instances" % type.__name__)
    return self.manager
```

所以 django 用 ManagerDescriptor 特意为 Manager 做的一层包装. 可以在 django.db.models.manager.py 中找到 ManagerDescriptor 的实现:

所要详述的是 __get__() 函数. python 的语法里有修饰器(descriptor)这么一说, 而 python 的属性类型就是这么实现的. descriptor 实现 __get__(), __set__(), 接着将其添加到一个类中. 譬如下面的例子:

```
class Celsius(object):
    def __init__(self, value=0.0):
        self.value= float(value)
    def __get__(self, instance, owner):
        print instance,owner
        return self.value
    def __set__(self, instance, value):
        print instance,value
        self.value= float(value)

class Temperature(object):
    celsius= Celsius()

t= Temperature()
t.celsius
Temperature.celsius
```

当对 descriptor 赋值的时候, 其本身 __set__ 会被调用, 取值的时候 __get__() 会被调用. __set__,__get__ 函数的 instance 参数即为类实例(所以, t.cellsius 调用 __get__() 的时候, instance 参数是 t, Temperature.celsius 调用 __get__() 的时候, instance 参数是 Temperature).

所以，可以通过判断 instance 来判断调用者是否是类实例. 也就由此可以拒绝类实例的访问，发现 ManagerDescriptor 就是这么实现的.

总结

Book.objects 实际上是一个 Manager，实际上的实际上却是一个 ManagerDescriptor，但真正起作用的还是 Manager, ManagerDescriptor 是修饰器，是 django 的保护技法.

从 Manager 的实现来看，它的多数函数会返回 QuerySet 对象，而且透漏了一个重点: QuerySet 对象可以看成是 Model 实例的集合.

我已经在 github 备份了 Django 源码的注释: Decode-Django，有兴趣的童鞋 fork 吧.

原文 http://www.cnblogs.com/daoluanxiaozi/p/3420559.html

在 Twitter，Netty 4 GC 开销降为五分之一

Netty 项目在7月份发布了 Netty 4的第一个版本，其性能的显著提升主要来源于垃圾收集开销的降低。在 Twitter，Netty 4经过完善已经获得了5倍的性能提升，但也有一些代价。

Netty 项目创始人和 Twitter 软件工程师 Trustin Lee 从2003年开始就一直编写网络应用程序框架。Netty 第一次公开发布是在2004年6月，其项目主页这样描述它，“一种异步事件驱动的网络应用程序框架，用于可维护的高性能协议服务器和客户端的快速开发。”

Lee 在博文“Netty 4在 Twitter：降低 GC 开销”中写道，Twitter 在许多地方使用 Netty 实现联网功能：

76 Finagle 是我们的协议无关的 RPC 系统，用于实现大部分内部服务，如搜索，它的传输层建立在 Netty 上。

77 TFE（Twitter 前端）是我们专有的填鸭式反向代理，它使用 Netty 为大部分面向公众的 HTTP 和 SPDY 流量提供服务。

78 Cloudhopper 每月用 Netty 向遍布世界各地数以百计的运营商发送数十亿条短信息。

Netty 包含一个反应堆模式的实现，它同时也是 Play 框架的核心。Play、Grails 和许多其它 Web 框架都采用一种 WAR-less Web 应用程序模式，它允许与底层 HTTP 服务器更紧密地集成。使用内部包含像 Netty 这种框架的服务器，异步编程会简单很多。异步编程和非阻塞 I/O 是“响应宣言（The Reactive Manifesto）”的核心。InfoQ 在“新兴趋势：响应式编程”中报道了这一新兴模式。

Netty 3使用 Java 对象表示 I/O 事件。Lee 谈道：

这样简单，但会产生大量的垃圾，尤其是在我们这样的规模下。Netty 4在新版本中对此做出了更改 ,取代生存周期短的事件对象 ,而以定义在生存周期长的通道对象上的方法处理 I/O 事件。它还有一个使用池的专用缓冲区分配器。

……每当收到新信息或者用户发送信息到远程端，Netty 3均会创建一个新的堆缓冲区。这意味着，对应每一个新的缓冲区，都会有一个‘new byte[capacity]’。这些缓冲区会导致 GC 压力，并消耗内存带宽：为了安全起见，新的字节数组分配时会用零填充，这会消耗内存带宽。然而，用零填充的数组很可能会再次用实际的数据填充，这又会消耗同样的内存带宽。如果 Java 虚拟机（JVM）提供了创建新字节数组而又无需用零填充的方式，那么我们本来就可以将内存带宽消耗减少50%，但是目前没有那样一种方式。

在 Netty 4中，代码定义了粒度更细的 API，用来处理不同的事件类型，而不是创建事件对象。它还实现了一个新缓冲池，那是一个纯 Java 版本的 jemalloc（Facebook 也在用）。现在，Netty 不会再因为用零填充缓冲区而浪费内存带宽了。不过，由于它不依赖于 GC，开发人员需要小心内存泄漏。如果忘记在处理程序中释放缓冲区，那么内存使用率会无限地增长。

这些变化没有向后兼容 Netty 3，但其垃圾生成速度是原来的1/5，而垃圾清理速度快了5倍。

Lee 写道：

我们比较了两个分别建立在 Netty 3和4基础上 echo 协议服务器。（Echo 非常简单，这样，任何垃圾的产生都是 Netty 的原因，而不是协议的原因）。我使它们服务于相同的分布式 echo 协议客户端，来自这些客户端的16384个并发连接重复发送256字节的随机负载，几乎使千兆以太网饱和。

根据测试结果，Netty 4：

79 GC 中断频率是原来的1/5：45.5 vs. 9.2次/分钟

80 垃圾生成速度是原来的1/5：207.11 vs 41.81 MiB/秒

Lee 提到，在 Twitter 中采用 Netty 4还有一些障碍，那就是缓冲区泄漏和核心复杂。该项目希望增加更多功能，包括 HTTP/2、异步 DNS 解析以及客户端 HTTP 和 SOCKS 代理支持。

Yahoo 工程部门有一篇类似的文章，内容是关于 Netty 如何帮助他们成倍地提升 Storm 集群的速度。在名为“Netty 让 Storm 飞速运行”的文章中，Bobby Evans 写道：

在雅虎，我们都是用自己的产品进行开发。在将 Netty 作为 Storm 集群的默认消息层之前，我需要一些数据来确认，它与当前的默认消息层 zeromq 相比怎么样。要做到这一点，我需要一个能够使 Storm 消息层达到极限的基准测试程序，因此，我写了一个。那是一个简单的高速测试，用于确认 Storm 在不同的 Bolt 和 Spout 之间推送消息有多快。它允许同时启动多个具有不同复杂度的 Topology 来发送定长消息。

Evans 指出，在小规模测试中（没有资源冲突），Netty 比 zeromq 更快（40-100%）。在大规模测试中，它也遇到了性能问题，但减少了解决问题的线程数。

对于大量短消息而言，Netty 的默认设置并不是很好，即使该节点上只有它自己在运行。但是，当把它限制在单线程上，我们每秒能够获得比 zeromq 多85%到111%的消息，之后网络再次饱和。

Evans 指出，Netty 现在是 Yahoo Storm 集群的默认消息层。

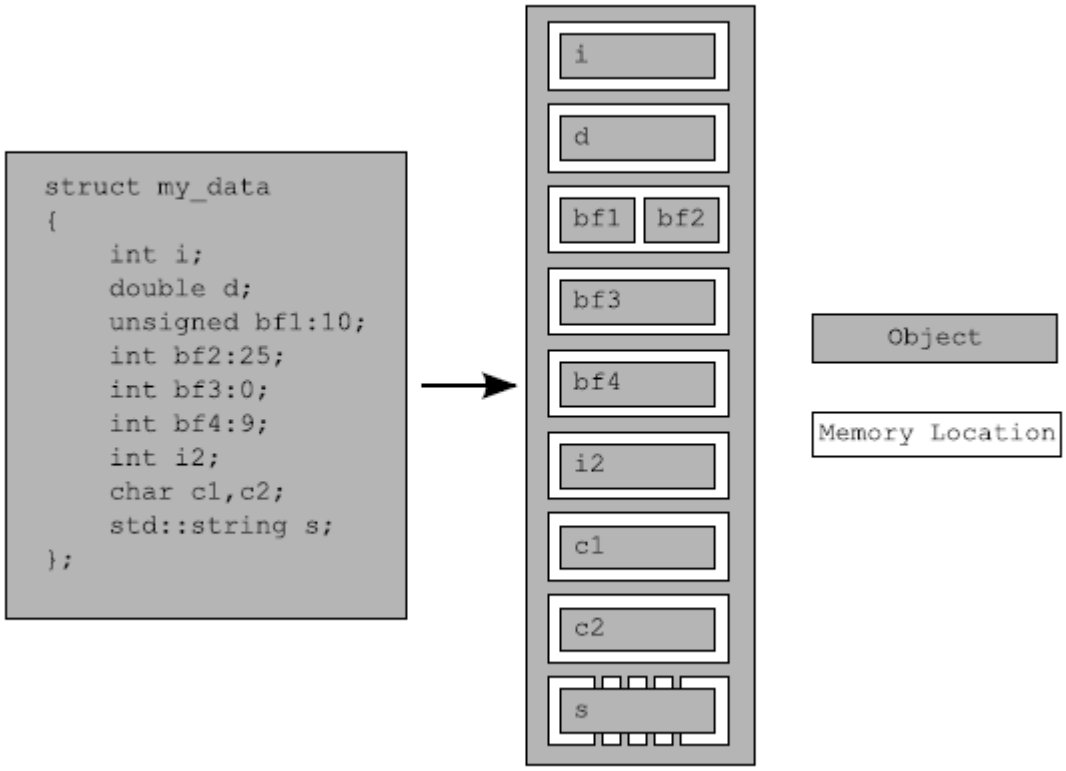
Netty 4的改进对许多开源项目都大有益处。该框架有一个长长的相关项目列表，包括 Akka、Apache James、HornetQ 和 Vert.x，这里仅举这几例。要了解更多关于 Netty 4的信息，请查看 netty.io 或 Lee 的博文。

原文 http://www.infoq.com/cn/news/2013/11/netty4-twitter

C++11 并发指南七(C++11 内存模型一：介绍)

第六章主要介绍了 C++11 中的原子类型及其相关的 API，原子类型的大多数 API 都需要程序员提供一个 std::memory_order（可译为内存序，访存顺序）的枚举类型值作为参数，比如 :atomic_store，atomic_load，atomic_exchange，atomic_compare_exchange 等 API 的最后一个形参为 std::memory_order order，默认值是 std::memory_order_seq_cst（顺序一致性）。那么究竟什么是 std::memory_order 呢，为了解答这个问题，我们先来讨论 C++11 的内存模型。

一般来讲，内存模型可分为静态内存模型和动态内存模型，静态内存模型主要涉及类的对象在内存中是如何存放的，即从结构(structural)方面来看一个对象在内存中的布局，以一个简单的例子为例（截图参考《C++ Concurrency In Action》 P105 ）：



上面是一个简单的 C++ 类(又称 POD: Plain Old Data，它没有虚函数，没有继承)，它在内存中的布局如图右边所示（对于复杂类对象的内存布局，请参考《深度探索 C++对象模型》一书）。

动态内存模型可理解为存储一致性模型，主要是从行为(behavioral)方面来看多个线程对同一个对象同时(读写)操作时(concurrency)所做的约束，动态内存模型理解起来稍微复杂一些，涉及了内存，Cache，CPU 各个层次的交互，尤其是在共享存储系统中，为了保证程序执行的正确性，就需要对访存事件施加严格的限制。

文献中常见的存储一致性模型包括顺序一致性模型，处理器一致性模型，弱一致性模型，释放一致性模型，急切更新释放一致性模型、懒惰更新释放一致性模型，域一致性模型以及单项一致性模型。不同的存储一致性模型对访存事件次序的限制不同，因而对程序员的要求和所得到的的性能也不一样。存储一致性模型对访存事件次序施加的限制越弱，我们就越有利于提高程序的性能，但编程实现上更困难。

顺序一致性模型由 Lamport 于 1979 年提出。顺序一致性模型最好理解但代价太大，原文指出：

... the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

该模型指出：如果在共享存储系统中多机并行执行的结果等于把每一个处理器所执行的指令流按照某种方式顺序地交织在一起在单机上执行的结果，则该共享存储系统是顺序一致性的。

顺序一致性不仅在共享存储系统上适用，在多处理器和多线程环境下也同样适用。而在多处理器和多线程环境下理解顺序一致性包括两个方面，(1). 从多个线程平行角度来看，程序最终的执行结果相当于多个线程某种交织执行的结果，(2)从单个线程内部执行顺序来看，该线程中的指令是按照程序事先已规定的顺序执行的(即不考虑运行时 CPU 乱序执行和 Memory Reorder)。

我们以一个具体的例子来理解顺序一致性：

情况 1	情况 2	情况 3	情况 4	情况 5	情况 6
a = 1;	b = 2;	a = 1;	a = 1;	b = 2;	b = 2;
R1 = b;	R2 = a;	b = 2;	b = 2;	a = 1;	a = 1;
b = 2;	a = 1;	R1 = b;	R2 = a;	R1 = b;	R2 = b;
R2 = a;	R1 = b;	R2 = a;	R1 = b;	R2 = a;	R1 = b;
R1 == 0, R2 == 1	R1 == 2, R2 == 0	R1 == 2, R2 == 1	R1 == 2, R2 == 1	R1 == 2, R2 == 1	R1 == 2, R2 ==

假设存在两个共享变量 a, b，初始值均为 0，两个线程运行不同的指令，如下表格所示，线程 1 设置 a 的值为 1,然后设置 R1 的值为 b,线程 2 设置 b 的值为 2,并设置 R2 的值为 a，请问在不加任何锁或者其他同步措施的情况下，R1，R2 的最终结果会是多少？

线程 1	线程 2
a = 1;	b = 2;
R1 = b;	R2 = a;

由于没有施加任何同步限制，两个线程将会交织执行，但交织执行时指令不发生重排，即线程 1 中的 a = 1 始终在 R1 = b 之前执行，而线程 2 中的 b = 2 始终在 R2 = a 之前执行 ，因此可能的执行序列共有 4!/(2!*2!) = 6 种：

上面的表格列举了两个线程交织执行时所有可能的执行序列，我们发现，R1,R2 最终结果只有 3 种情况,分别是 R1 == 0, R2 == 1(情况 1) ,R1 == 2, R2 == 0(情况2) 和 R1 == 2，R2 == 1(情况 3， 4， 5，6)。结合上面的例子，我想大家应该理解了什么是顺序一致性。

因此，多线程环境下顺序一致性包括两个方面，(1). 从多个线程平行角度来看，程序最终的执行结果相当于多个线程某种交织执行的结果，(2)从单个线程内部执行顺序来看，该线程

中的指令是按照程序事先已规定的顺序执行的(即不考虑运行时 CPU 乱序执行和

Memory Reorder)。

当然，顺序一致性代价太大，不利于程序的优化，现在的编译器在编译程序时通常将指令重新排序（当然前提是保证程序的执行结果是正确的），例如，如果两个变量读写互不相关，

编译器有可能将读操作提前（暂且称为预读 prefetch 吧），或者尽可能延迟写操作，假设

如下面的代码段：



```
int a = 1, b = 2;

void func()
{
      a = b + 22;
      b = 22;
}
```



在 GCC 4.4 (X86-64)编译条件下，优化选项为 -O0 时，汇编后关键代码如下：

```
movl      b(%rip), %eax ; 将 b 读入 %eax

addl      $22, %eax ; %eax 加 22, 即 b + 22

movl      %eax, a(%rip) ; % 将 %eax 写回至 a, 即 a = b + 22

movl      $22, b(%rip) ; 设置 b = 22
```

而在设置 -O2 选项时，汇编后的关键代码如下：

```
movl      b(%rip), %eax ; 将 b 读入 %eax

movl      $22, b(%rip) ; b = 22

addl      $22, %eax ; %eax 加 22

movl      %eax, a(%rip) ; 将 b + 22 的值写入 a,即 a = b + 2
```

由上面的例子可以看出，编译器在不同的优化级别下确实对指令进行了不同程度重排，在 -O0(不作优化)的情况下，汇编指令和 C 源代码的逻辑相同，但是在 -O2 优化级别下，汇编指令和原始代码的执行逻辑不同，由汇编代码可以观察出，b = 22 首先执行，最后才是 a = b + 2， 由此看出，编译器会根据不同的优化等级来适当地对指令进行重排。在单线程条件下上述指令重排不会对执行结果带来任何影响，但是在多线程环境下就不一定了。如果另外一个线程依赖 a，b 的值来选择它的执行逻辑，那么上述重排将会产生严重问题。编译器优化是一门深奥的技术，但是无论编译器怎么优化，都需要对优化条件作出约束，尤其是在多线程条件下，不能无理由地优化，更不能错误地优化。

另外，现代的 CPU 大都支持多发射和乱序执行，在乱序执行时，指令被执行的逻辑可能和程序汇编指令的逻辑不一致，在单线程条件下，CPU 的乱序执行不会带来大问题，但是在多核多线程时代，当多线程共享某一变量时，不同线程对共享变量的读写就应该格外小心，不适当的乱序执行可能导致程序运行错误。因此，CPU 的乱序执行也需要作出适当的约束。

综上所述，我们必须对编译器和 CPU 作出一定的约束才能合理正确地优化你的程序，那么这个约束是什么呢？答曰：内存模型。C++程序员要想写出高性能的多线程程序必须理解内存模型，编译器会给你的程序做优化(静态)，CPU 为了提升性能也有乱序执行(动态)，总之，程序在最终执行时并不会按照你之前的原始代码顺序来执行，因此内存模型是程序员、编译器，CPU 之间的契约，遵守契约后大家就各自做优化，从而尽可能提高程序的性能。

C++11 中规定了 6 中访存次序(Memory Order)，如下：

```
enum memory_order {
    memory_order_relaxed,
    memory_order_consume,
    memory_order_acquire,
    memory_order_release,
    memory_order_acq_rel,
```

```
    memory_order_seq_cst
};
```



std::memory_order 规定了普通访存操作和相邻的原子访存操作之间的次序是如何安排的，在多核系统中，当多个线程同时读写多个变量时，其中的某个线程所看到的变量值的改变顺序可能和其他线程写入变量值的次序不相同。同时，不同的线程所观察到的某变量被修改次序也可能不相同。然而，如果保证所有对原子变量的操作都是顺序的话，可能对程序的性能影响很大，因此，我们可以通过 std::memory_order 来指定编译器对访存次序所做的限制。因此，在原子类型的 API 中，我们可以通过额外的参数指定该原子操作的访存次序(内存序)，默认的内存序是 **std::memory_order_seq_cst**。

我们可以把上述 6 中访存次序(内存序)分为 3 类，顺序一致性模型(std::memory_order_seq_cst)，Acquire-Release 模型(std::memory_order_consume, std::memory_order_acquire, std::memory_order_release, std::memory_order_acq_rel,)和 Relax 模型(std::memory_order_relaxed)。三种不同的内存模型在不同类型的 CPU 上(如 X86，ARM，PowerPC 等)所带来的代价也不一样。例如，在 X86 或者 X86-64平台下，Acquire-Release 类型的访存序不需要额外的指令来保证原子性，即使顺序一致性类型操作也只需要在写操作(Store)时施加少量的限制，而在读操作(Load)则不需要花费额外的代价来保证原子性。

========== TL;DR ==========

附：本文剩余部分将介绍其他的存储器一致模型中的其他几种较常见的模型：处理器一致性(Processor Consistency)模型，弱一致性(Weak Consistency)模型，释放一致性(Release Consistency)模型。[注：以下内容来自中国科学院计算技术研究所胡伟武老师写的《计算机体系结构》(清华大学出版社)，该书是胡伟武老师给研究生讲课所用的教材，本文略有删改]

处理器一致性(Processor Consistency)模型：处理器一致性(Processor Consistency)模型比顺序一致性模型弱，因此对于某些在顺序一致性模型下能够正确执行的程序在处理器一致性条件下执行时可能会导致错误的结果，处理器一致性模型对访存事件发生次序施加的限制是：(1). 在任意读操作(Load)被允许执行之前，所有在同一处理器中先于这一 Load 的读操作都已完成；(2). 在任意写操作(Store)被允许执行之前，所有在同一处理器中先于这一 Store 的访存操作(包括 Load 和 Store 操作)都已完成。上述条件允许 Store 之后的 Load 越过 Store 操作而有限执行。

弱一致性(Weak Consistency)模型：弱一致性(Weak Consistency)模型的主要思想是将同步操作和普通的访存操作区分开来，程序员必须用硬件可识别的同步操作把对可写共享单元的访存保护起来，以保证多个处理器对可写单元的访问是互斥的。弱一致性对访存事件发生次序的限制如下：(1). 同步操作的执行满足顺序一致性条件; (2). 在任一普通访存操作被允许执行之前，所有在同一处理器中先于这一访存操作的同步操作都已完成; (3). 在任一同步操作被允许执行之前，所有在同一处理器中先于这一同步操作的普通操作都已完成。上述条件允许在同步操作之间的普通访存操作执行时不用考虑进程之间的相关，虽然弱一致性增加了程序员的负担，但是它能有效地提高系统的性能。

释放一致性(Release Consistency)模型：释放一致性(Release Consistency)模型是对弱一致性(Weak Consistency)模型的改进，它把同步操作进一步分成了获取操作(Acquire)和释放操作(Release)。Acquire 用于获取对某些共享变量的独占访问权，而 Release 则用于释放这种访问权，释放一致性(Release Consistency)模型访存事件发生次序的限制如下：(1). 同步操作的执行满足顺序一致性条件; (2). 在任一普通访存操作被允许执行之前，所有在同一处理器中先于这一访存操作的 Acquire 操作都已完成; (3). 在任一 Release 操作被允许执行之前，所有在同一处理器中先于这一 Release 操作的普通操作都已完成。

在硬件实现的释放一致性模型中，对共享单元的访存是及时进行的，并在执行获取操作(Acquire)和释放操作(Release)时对齐。在共享虚拟存储系统或者在由软件维护的数据一致性的共享存储系统中，由于通信和数据交换的开销很大，有必要减少通信和数据交换的次数。为此，人们在释放一致性(Release Consistency)模型的基础上提出了急切更新释放一致性模型(Eager Release Consistency)和懒惰更新释放一致性模型(Lazy Release Consistency)。在急切更新释放一致性模型中，在临界区内的多个存数操作对共享内存的更新不是及时进行的，而是在执行 Release 操作之前(即退出临界区之前)集中进行，把多个存数操作合并在一起统一执行，从而减少了通信次数。而在懒惰更新释放一致性模型中，由一个处理器对某单元的存数操作并不是由此处理器主动传播到所有共享该单元的其他处理器，而是在其他处理器要用到此处理器所写的数据时(即其他处理器执行 Acquire 操作时)再向此处理器索取该单元的最新备份，这样可以进一步减少通信量。

==========

好了，本文主要介绍了内存模型的相关概念，并重点介绍了顺序一致性模型（附带介绍了几种常见的存储一致性模型），并以一个实际的小例子向大家介绍了为什么程序员需要理解内存模型，总之，C++ 程序员要想写出高性能的多线程程序必须理解内存模型，因为编译器会给你的程序做优化(如指令重排等)，CPU 为了提升性能也有多发射和乱序执行，因此程序在最终执行时并不会按照你之前的原始代码顺序来执行，所以内存模型是程序员、编译器，CPU 之间的契约，遵守契约后大家就各自做优化，从而尽可能提高程序的性能。

下一节我将给大家介绍 C++11 内存模型中的 6 种访存次序(或内存序)(std::memory_order_relaxed, std::memory_order_consume, std::memory_order_acquire, std::memory_order_release, std::memory_order_acq_rel, std::memory_order_seq_cst)各自的意义以及常见的用法，希望感兴趣的同学继续关注，如果您发现文中的错误，一定尽快告诉我 ;-)

另外，后续的几篇博客我会给大家介绍更多的与内存模型相关的知识，我在 Github 上维护了一个页面，主要是与内存模型相关资料的链接，感兴趣的同学可以参考里面的资料自己阅读。

原文 http://www.cnblogs.com/haippy/p/3412858.html

Meta Programming 主要解决什么了问题？

> Metaprogramming is the writing of computer programs that write or manipulate other programs (or themselves) as their data, or that do part of the work at compile time that would otherwise be done at runtime. In some cases, this allows programmers to minimize the number of lines of code to express a solution (hence reducing development time), or it gives programs greater flexibility to efficiently handle new situations without recompilation.

没有接触过 meta programming, 感觉"manipulate other programs (or themselves) as their data" 这个说法有点虚, 能否给一些确切的例子说明其妙用之处?

另外, 为什么说 LISP 与 Ruby 在 meta programming 上比 Python 做得要好?

原则上说，目前能看到的 Meta Programming 有两种，第一种专注于重新构成语法，即 DSL，Lisp/Scheme 属于此类，这几年的新科状元 Ruby 也属于这一派，甚至还有走得更远但相对小众的 REBOL Language；第二种专注于在编译期进行计算和分派以达到针对特定场景的自动代码选择优化，C++ 的模板属于此类。C 的宏则兼有两者，但能力都比较弱。据我见过的情况，学院派的比较认可前一种，后一种则借 C++ 的东风用得更广泛。

问题引言里的那句话，我认为是在说 C++，因为它强调了 do part of the work at compile time。但 Python 的 Meta-programming，我闻所未闻。Python 1.2 开始用到现在，我确实没有听说 Python 有什么构建 DSL 的能力，更遑论所谓 Meta-programming。Python 社区的文化，也

向来没有在语法上玩花样的传统。于是大略查了查，它说的是应该是第三项东西：类结构动态调整，本质来说，就是反射（reflection）。参照 DeveloperWorks 上的这一篇：Metaclass programming in Python，以及这一篇：Metaprogramming。不难看出，这两篇文章谈的是完全另一个话题。因此谈所谓 Lisp 和 Ruby 在 Meta-programming 上为什么比 Python 做得好，在我看来是一个伪命题，因为两者说的 Meta-programming 的内涵和外延都不一样。

在实际工作中，无论哪一种 Meta-programming，我都持否定态度。但理由和前面朋友的观点不同。我认为 Meta-programming 破坏了交流中的一项基本原则：同样的编程语言，应当有同样的交流习惯。过于随意地构造 DSL，让程序员之间的交流变得更加困难。这一方面的典型例子是 REBOL，几种 DSL 语法差异明显，重新学习的成本过高，也实际上助长了社区的分裂倾向。闲来无事，自己玩玩也就算了；放在工作里，害人害己。

至于 Lisp/Scheme 家族的 Meta Programming，与其说是做得更好，倒不如说是语法结构过于简单导致其怎么变换都是长得一个样子。关于这一点，不妨对比一下 REBOL，它构造出来的几个 DSL 语法上没有多少相似之处。不过，对我来说，Lisp/Scheme 这种语法简单其实才是我喜欢的 Scheme：编译器的设计者可以把更多的精力花在运行时和优化上。

还是那句话，我们写程序是为了解决问题。抽象和表达方式是手段，不是目的。汲汲于语法上的花头，是耍滑头。

原文 http://www.zhihu.com/question/21486259/answer/18408267

Qt 学习之路 2（70）：进程间通信

我们将进程理解为相互独立的正在运行的程序。由于二者是相互独立的，就存在交互的可能性，也就是我们所说的进程间通信（Inter-Process Communication，IPC）。不过也正因此，

我们的一些简单的交互方式，比如普通的信号槽机制等，并不适用于进程间的相互通信。我们说过，进程是操作系统的基本调度单元，因此，进程间交互不可避免与操作系统的实现息息相关。

Qt 提供了四种进程间通信的方式：

81 使用共享内存（shared memory）交互：这是 Qt 提供的一种各个平台均有支持的进程间交互的方式。

82 TCP/IP：其基本思想就是将同一机器上面的两个进程一个当做服务器，一个当做客户端，二者通过网络协议进行交互。除了两个进程是在同一台机器上，这种交互方式与普通的 C/S 程序没有本质区别。Qt 提供了 QNetworkAccessManager 对此进行支持。

83 D-Bus：freedesktop 组织开发的一种低开销、低延迟的 IPC 实现。Qt 提供了 QtDBus 模块，把信号槽机制扩展到进程级别（因此我们前面强调是“普通的”信号槽机制无法实现 IPC），使得开发者可以在一个进程中发出信号，由其它进程的槽函数响应信号。

84 QCOP（Qt COmmunication Protocol）：QCOP 是 Qt 内部的一种通信协议，用于不同的客户端之间在同一地址空间内部或者不同的进程之间的通信。目前，这种机制只用于 Qt for Embedded Linux 版本。

从上面的介绍中可以看到，通用的 IPC 实现大致只有共享内存和 TCP/IP 两种。后者我们前面已经大致介绍过（应用程序级别的 QNetworkAccessManager 或者更底层的 QTcpSocket 等）；本章我们主要介绍前者。

Qt 使用 QSharedMemory 类操作共享内存段。我们可以把 QSharedMemory 看做一种指针，这种指针指向分配出来的一个共享内存段。而这个共享内存段是由底层的操作系统提供，可

以供多个线程或进程使用。因此，QSharedMemory 可以看做是专供 Qt 程序访问这个共享内存段的指针。同时，QSharedMemory 还提供了单一线程或进程互斥访问某一内存区域的能力。当我们创建了 QSharedMemory 实例后，可以使用其 create() 函数请求操作系统分配一个共享内存段。如果创建成功（函数返回 true），Qt 会自动将系统分配的共享内存段连接（attach）到本进程。

前面我们说过，IPC 离不开平台特性。作为 IPC 的实现之一的共享内存也遵循这一原则。有关共享内存段，各个平台的实现也有所不同：

- Windows：QSharedMemory 不“拥有”共享内存段。当使用了共享内存段的所有线程或进程中的某一个销毁了 QSharedMemory 实例，或者所有的都退出，Windows 内核会自动释放共享内存段。
- Unix：QSharedMemory “拥有”共享内存段。当最后一个线程或进程同共享内存分离，并且调用了 QSharedMemory 的析构函数之后，Unix 内核会将共享内存段释放。注意，这里与 Windows 不同之处在于，如果使用了共享内存段的线程或进程没有调用 QSharedMemory 的析构函数，程序将会崩溃。
- HP-UX：每个进程只允许连接到一个共享内存段。这意味着在 HP-UX 平台，QSharedMemory 不应被多个线程使用。

下面我们通过一段经典的代码来演示共享内存的使用。这段代码修改自 Qt 自带示例程序（注意这里直接使用了 Qt5，Qt4 与此类似，这里不再赘述）。程序有两个按钮，一个按钮用于加载一张图片，然后将该图片放在共享内存段；第二个按钮用于从共享内存段读取该图片并显示出来。

```
//!!! Qt5
```

```
class QSharedMemory;


class MainWindow : public QMainWindow
{
    Q_OBJECT


public:
    MainWindow(QWidget *parent = 0);
    ~MainWindow();


private:
    QSharedMemory *sharedMemory;
};
```

头文件中，我们将 MainWindow 添加一个 sharedMemory 属性。这就是我们的共享内存段。

接下来得实现文件中：

```
const char *KEY_SHARED_MEMORY = "Shared";


MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent),
      sharedMemory(new QSharedMemory(KEY_SHARED_MEMORY, this))
{
    QWidget *mainWidget = new QWidget(this);
    QVBoxLayout *mainLayout = new QVBoxLayout(mainWidget);
    setCentralWidget(mainWidget);
```

```
QPushButton *saveButton = new QPushButton(tr("Save"), this);
mainLayout->addWidget(saveButton);
QLabel *picLabel = new QLabel(this);
mainLayout->addWidget(picLabel);
QPushButton *loadButton = new QPushButton(tr("Load"), this);
mainLayout->addWidget(loadButton);
```

构造函数初始化列表中我们将 sharedMemory 成员变量进行初始化。注意我们给出一个键（Key），前面说过，我们可以把 QSharedMemory 看做是指向系统共享内存段的指针，而这个键就可以看做指针的名字。多个线程或进程使用同一个共享内存段时，该键值必须相同。接下来是两个按钮和一个标签用于界面显示，这里不再赘述。

下面来看加载图片按钮的实现：

```
connect(saveButton, &QPushButton::clicked, [=]() {
    if (sharedMemory->isAttached()) {
        sharedMemory->detach();
    }
    QString filename = QFileDialog::getOpenFileName(this);
    QPixmap pixmap(filename);
    picLabel->setPixmap(pixmap);

    QBuffer buffer;
    QDataStream out(&buffer);
    buffer.open(QBuffer::ReadWrite);
    out << pixmap;
```

```
        int size = buffer.size();
        if (!sharedMemory->create(size)) {
            qDebug() << tr("Create Error: ") << sharedMemory->errorString();
        } else {
            sharedMemory->lock();
            char *to = static_cast(sharedMemory->data());
            const char *from = buffer.data().constData();
            memcpy(to, from, qMin(size, sharedMemory->size()));
            sharedMemory->unlock();
        }
    });
```

点击加载按钮之后，如果 sharedMemory 已经与某个线程或进程连接，则将其断开（因为我们就要向共享内存段写入内容了）。然后使用 QFileDialog 选择一张图片，利用 QBuffer 将图片数据作为 char * 格式。在即将写入共享内存之前，我们需要请求系统创建一个共享内存段（QSharedMemory::create() 函数），创建成功则开始写入共享内存段。需要注意的是，在读取或写入共享内存时，都需要使用 QSharedMemory::lock() 函数对共享内存段加锁。共享内存段就是一段普通内存，所以我们使用 C 语言标准函数 memcpy() 复制内存段。不要忘记之前我们对共享内存段加锁，在最后需要将其解锁。

接下来是加载按钮的代码：

```
    connect(loadButton, &QPushButton::clicked, [=]() {
        if (!sharedMemory->attach()) {
            qDebug() << tr("Attach Error: ") << sharedMemory->errorString();
        } else {
```

```
            QBuffer buffer;
            QDataStream in(&buffer);
            QPixmap pixmap;
            sharedMemory->lock();
            buffer.setData(static_cast(sharedMemory->constData()), sharedMemory->size());
            buffer.open(QBuffer::ReadWrite);
            in >> pixmap;
            sharedMemory->unlock();
            sharedMemory->detach();
            picLabel->setPixmap(pixmap);
        }
    });
```

如果共享内存段已经连接，还是用 QBuffer 读取二进制数据，然后生成图片。注意我们在操作共享内存段时还是要先加锁再解锁。最后在读取完毕后，将共享内存段断开连接。

注意，如果某个共享内存段不是由 Qt 创建的，我们也是可以在 Qt 应用程序中使用。不过这种情况下我们必须使用 QSharedMemory::setNativeKey() 来设置共享内存段。使用原始键（native key）时，QSharedMemory::lock() 函数就会失效，我们必须自己保护共享内存段不会在多线程或进程访问时出现问题。

IPC 使用共享内存通信是一个很常用的开发方法。多个进程间得通信要比多线程间得通信少一些，不过在某一族的应用情形下，比如 QQ 与 QQ 音乐、QQ 影音等共享用户头像，还是非常有用的。

原文 http://www.devbean.net/2013/11/qt-study-road-2-ipc/

iOS 7教程：浅析 Text Kit

在 iOS7中，苹果引入了 Text Kit--Text Kit 是一个快速而又现代化的文字排版和渲染引擎。

Text Kit 在 UIKit framework 中的定义了一些类和相关协议，它最主要的作用就是为程序提供文字排版和渲染的功能。

Text Kit 是 iOS 7中引入的一个新功能，非常值得开发者使用，下面先看看本文的目录结构：

什么是 Text Kit
Text Kit 架构
Text Kit 特点
Text Kit 功能概述
Text Kit 中重要的一些对象
Text Kit 示例
小结
推荐 Text Kit 学习资源

什么是 Text Kit

在 iOS7中，苹果引入了 Text Kit--Text Kit 是一个快速而又现代化的文字排版和渲染引擎。

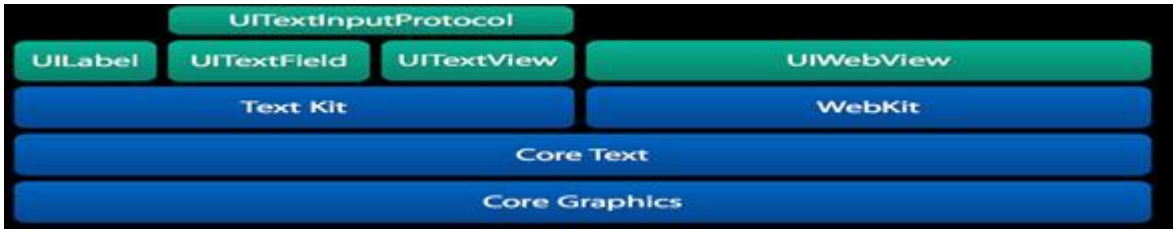
Text Kit 在 UIKit framework 中的定义了一些类和相关协议，它最主要的作用就是为程序提供文字排版和渲染的功能。在程序中，通过 Text Kit 可以对文字进行存储(store)、布局(lay out)，以及用最精细的排版方式(例如文字间距、换行和对齐等)来显示文本内容。 苹果引入 Text Kit 的目的并非要取代已有的 Core Text，Core Text 的主要作用也是用于文字的排版和渲染中，它是一种先进而又处于底层技术，如果我们需要将文本内容直接渲染到图形上下文(Graphics context)时，从性能和易用性来考虑，最佳方案就是使用 Core Text。而如果我们直

接利用苹果提供的一些控件(例如 UITextView、UILabel 和 UITextField 等)对文字进行排版，无疑就是借助于 UIkit framework 中 Text Kit 提供的 API。

Text Kit 架构

下面，我们通过图1(此图来自 WWDC2013 Session 210)来了解一下 Text Kit 的架构。图1是基于 iOS 7绘制的 ,从图中 ,我们可以看到 Text Kit 是基于 Core Text 构建的 ,它通过 Core Text 与 Core Graphics 进行交互。而 UI 控件(UILabel、UITextField 和 UITextView)则构建于 Text Kit 之上，可见这些文本控件可以利用 Text Kit 提供的 API 来对文字进行排版和渲染处理。 从图中我们也可以看到 SDK 提供的 UIWebView 是基于 WebKit 的，它不能使用 Text Kit 提供的功能。

图一 的功能



我们再来看看图1中的相关组件在 iOS6里面是如何对应的，如图2所示，可以看出在 iOS 6 中是没有 Text Kit，并且 UILabel、UIText 和 UITextView 是基于 String Drawing 和 WebKit 构建的。其中 String Drawing 是与 Core Graphics 直接通讯。

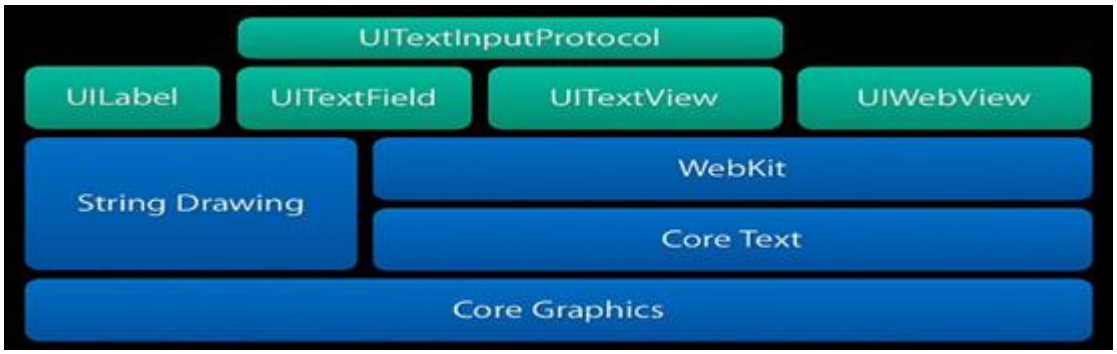


图2 在 iOS 6中并没有 Text Kit

Text Kit 特点

从上面的介绍中，我们可以了解到 Text Kit 在 UIKit 中的作用非常重要。Text Kit 在实际开发中具有如下特点：

1.在 UI 控件中 Text Kit 完全掌控着文字的排版和渲染

2.UITextView、UITextField 和 UILabel 是构建于 Text Kit 之上的

3.能够与动画、UICollectionView 和 UITableView 做到无缝集成

4.Text Kit 具有这样一些能力：Subclassing、Delegation 和 Notifcation。

Text Kit 功能概述

下面我们看看通过 Text Kit，都能实现那些功能(这里列出了是一些常用和重要功能)：

1.对文字进行分页或多列排版

2.支持文字的换行、折叠和着色等处理

3.可以调整字与字之间的距离、行间距、文字大小、指定特定的字体

4.支持富文本编辑，可以自定义文字截断

5.支持凸版印刷效果(letterpress)

6.支持数据类型的检测(例如链接、附件等)

如图3，是利用 Text Kit 对文字做的分页排版

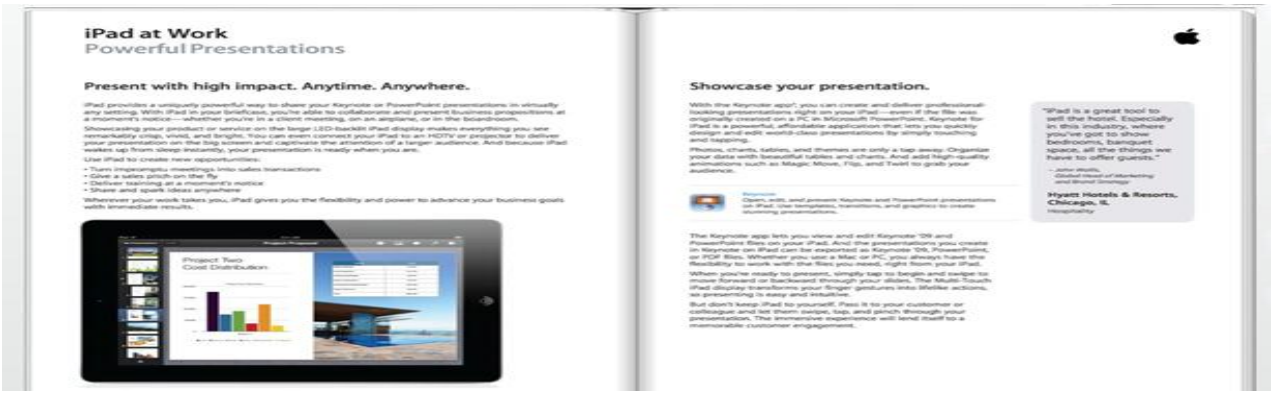


图3 利用 Text Kit 做的分页排版效果

再看图4，是利用 Text Kit 做的换行处理，其中对某个路径范围做了排除。



图4 利用 Text Kit 做的换行处理效果

再来看看利用 Text Kit 做的凸版印刷效果，如图5所示



图5 利用 Text Kit 做的凸版印刷效果

Text Kit 中重要的一些对象

下面我们来看看 Text Kit 中重要的几个对象。

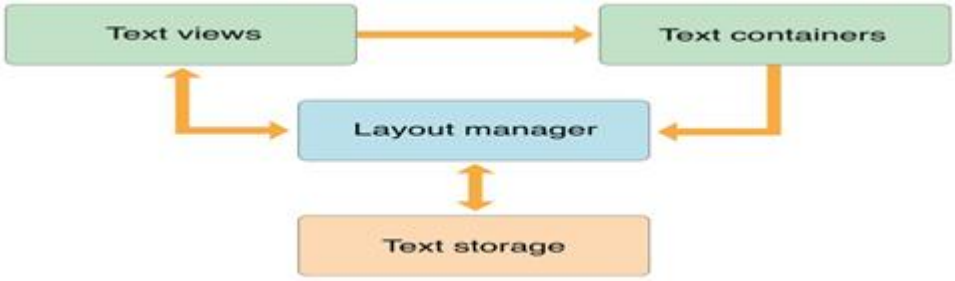


图6 Text Kit 中重要的几个对象

如图6所示，Text Kit 中主要有4个重要的对象。

1.Text View 是用来显示文本内容的控件，主要包括 UILabel、UITextView 和 UITextField。

2.Text containers 对应着 NSTextContainer 类。NSTextContainer 定义了文本可以排版的区域。一般来说，都是矩形区域，当然，也可以根据需求，通过子类化 NSTextContainer 来创建别的一些形状，例如圆形、不规则的形状等。NSTextContainer 不仅可以创建文本可以填充的区域，它还维护着一个数组——该数组定义了一个区域，排版的时候文字不会填充该区域，因此，我们可以在排版文字的时候，填充非文本元素(例如图片，如图4所示)。

3.Layout manager 对应着 NSLayoutManager 类。该类负责对文字进行编辑排版处理——通过将存储在 NSTextStorage 中的数据转换为可以在视图控件中显示的文本内容，并把统一的字符编码映射到对应的字形(glyphs)上，然后将字形排版到 NSTextContainer 定义的区域中。

4.Text storage 对应着 NSTextStorage 类。该类定义了 Text Kit 扩展文本处理系统中的基本存储机制。NSTextStorage 继承自 NSmutableAttributedString，主要用来存储文本的字符和相关属性。另外，当 NSTextStorage 中的字符或属性发生了改变，会通知 NSLayoutManager，进而做到文本内容的显示更新。

通常情况下，NSTextStorage、NSLayoutManager 和 NSTextContainer 是一一对应的。如图7所示关系：



图7 普通排版

当然，如果需要将文字显示为多列，或多页，可以按照如图8所示关系——使用多个NSTextContainer。



图8 多页或者多列排版

如果针对不同的排版方式，则可以使用多个 NSLayoutManager，如图9所示



图9 不同的排版方式

如图10所示，通过形象的方式，对 UITextView 的组成做了分解。通常，我们在设备上只能看到最右边的文本显示界面，而内部的 NSTextStorage、NSLayoutManager 和 NSTextContainer 是看不出来的。通常由 NSLayoutManager 从 NSTextStorage 中读取出文本数据，然后根据一定的排版方式，将文本排版到 NSTextContainer 中，再由 NSTextContainer 结合 UITextView 将最终效果显示出来。

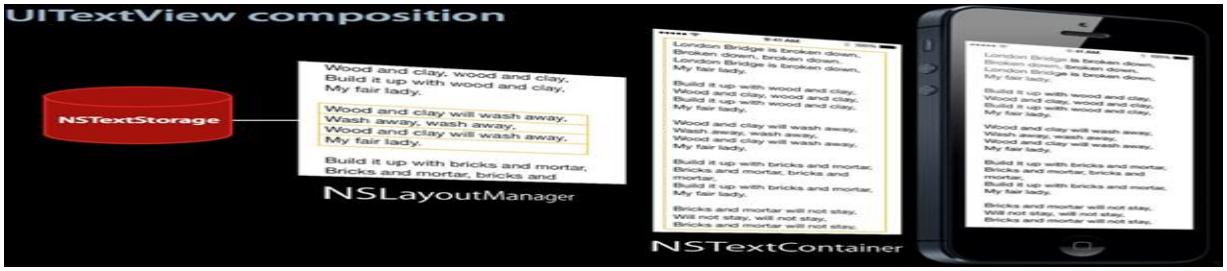


图10 UITextView 的分解

Text Kit 示例

前面对 Text Kit 做了一些介绍，下面我们配合一个例子(图文排版)，来进一步加深对 Text Kit 的认识。具体实现步骤如下：

1.打开 Xcode 5，新建一个 Single View Application 模板的程序，将工程命名为 ExclusionPath。

2.打开 Main.storyboard 文件，然后再默认 View Controller 的 View 里面分别添加一个 UITextView 和 UIImageView。并将这两个控件连接到 ViewController.h 中(名称分别为 textView 何 imageView)。然后给 textView 设置一些字符串，imageView 设置一个图片。

3.打开 ViewController.m 文件，找到 viewDidLoad 方法，用如下代码替换该方法：

```
85 - (void)viewDidLoad
86 {
87     [super viewDidLoad];
88 
89     //创建一个平移手势对象，该对象可以调用 imagePanned：方法
90     UIPanGestureRecognizer *panGes = [[UIPanGestureRecognizer alloc] initWithTarget:self action:@selector(imagePanned:)];
91     [self.imageView addGestureRecognizer:panGes];
92 
93     self.textView.textContainer.exclusionPaths = @[[self translatedBezierPath]];
94 }
```

在上面的代码中，给 imageView 添加了一个平移手势。另外通过调用 translatedBezierPath 方法，给 textView 的 textContainer 设置 exclusionPaths 属性值。表示需要排除的区域（也就是图片在排版中显示的位置）。

下面来看一下 translatedBezierPath 方法的实现，如下代码所示

```
95 - (UIBezierPath *)translatedBezierPath
96 {
97     CGRect butterflyImageRect = [self.textView convertRect:self.imageView.frame fromView:self.view];
```

```
98      UIBezierPath     *newButterflyPath     =     [UIBezierPath
bezierPathWithRect:butterflyImageRect];
99
100     return newButterflyPath;
101 }
```

在上面的代码中，利用 imageView 的 frame 属性创建了一个 UIBezierPath ,然后将该值返回。

5. 还记得第3步中创建的平移手势吗。里面有一个 action 需要实现 imagePanned:，下面来看看这个方法的实现：

```
102 - (void)imagePanned:(id)sender
103 {
104     if ([sender isKindOfClass:[UIPanGestureRecognizer class]])
{
105         UIPanGestureRecognizer *localSender = sender;
106
107         if (localSender.state == UIGestureRecognizerStateBegan)
{
108             self.gestureStartingPoint       =       [localSender
translationInView:self.textView];
109             self.gestureStartingCenter = self.imageView.center;
110         }       else       if       (localSender.state       ==
UIGestureRecognizerStateChanged) {
111             CGPoint      currentPoint      =      [localSender
translationInView:self.textView];
112
113             CGFloat     distanceX     =     currentPoint.x     -
self.gestureStartingPoint.x;
114             CGFloat     distanceY     =     currentPoint.y     -
self.gestureStartingPoint.y;
115
116             CGPoint newCenter = self.gestureStartingCenter;
117
118             newCenter.x += distanceX;
119             newCenter.y += distanceY;
120
121             self.imageView.center = newCenter;
122
123             self.textView.textContainer.exclusionPaths       =
@[[self translatedBezierPath]];
124         }       else       if       (localSender.state       ==
UIGestureRecognizerStateEnded) {
```

```
125             self.gestureStartingPoint = CGPointZero;
126             self.gestureStartingCenter = CGPointZero;
127         }
128     }
129 }
```

在上面的代码中首先根据平移的距离来设置 imageView 的位置，然后利用 translatedBezierPath 方法重新计算了一下排除区域。 6. 至此代码编写完毕，下面来运行程序，看看实际效果。如图11所示：

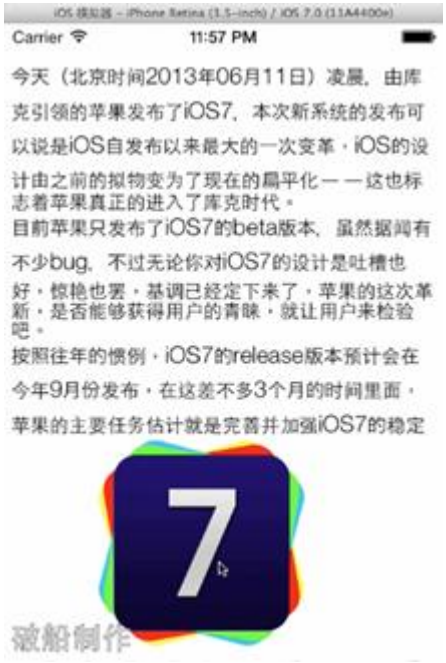


图11 运行效果

小结

实际上，上面的示例，只是揭秘了 Text Kit 功能的冰山一角。从 iOS7及以后的版本中，Text Kit 在 UIKit framework 里面占据重要的地位，Text Kit 在文字处理方面，具有非常强大的功能，并且开发者可以对 Text Kit 进行定制和扩展。据悉，苹果利用了2年的时间来开发 Text Kit，相信这对许多开发者来说都是福音。

推荐 Text Kit 学习资源

更多关于 Text Kit 的学习资料，请参考下面的内容：

wwdc 视频:

Introducing Text Kit

Advanced Text Layouts and Effects with Text Kit

Using Fonts with Text Kit

苹果官方参考文档

Text Programming Guide for iOS.pdf

NSLayoutManager Class Reference for iOS.pdf

NSLayoutManagerDelegate Protocol Reference for iOS.pdf

NSTextContainer Class Reference for iOS.pdf

NSTextStorage Class Reference for iOS.pdf

NSTextStorageDelegate Protocol Reference for iOS.pdf

苹果官方示例：

IntroToTextKit

原文：http://www.cocoachina.com/applenews/devnews/2013/1113/7342.html

key-value 存储系统 RocksDB

RocksDB 是一个来自 facebook 的可嵌入式的支持持久化的 key-value 存储系统，也可作为 C/S 模式下的存储数据库，但主要目的还是嵌入式。RocksDB 基于 LevelDB 构建。关于 RocksDB 的性能说明。

```
#include <assert>
#include "rocksdb/db.h"
rocksdb::DB* db;
rocksdb::Options options;
options.create_if_missing = true;
rocksdb::Status status =
  rocksdb::DB::Open(options, "/tmp/testdb", &db);
assert(status.ok());
...
```

下载：

https://github.com/facebook/rocksdb/issues/10

https://github.com/facebook/rocksdb#readme

原文 http://www.oschina.net/p/rocksdb

NoSQL 数据库面面观

Alexey Vasiliev 是一位知名的 Web 开发者与 Linux 系统管理员，曾参与开发过多个项目，如 falcon、mongodb_logger、sht_rails 及 piro 等项目。近日，Vasiliev 就当前各种 NoSQL 数据库的优势与劣势撰文进行了详尽的分析。这些分析与比较将会对广大开发者项目的 NoSQL 数据库选型提供一定的帮助与指导作用。

NoSQL 数据库现在已经变得非常流行了，在 NoSQL 这个大概念下实际上包含了大量的方式与项目，旨在实现各种数据库模型，他们与传统的关系型数据库管理系统存在着非常大的差别，而传统的关系型数据库系统是通过 SQL 的方式来访问数据的。在 NoSQL 领域中，传统观念中的模式可以通过不同的数据结构来实现，如散列表、数组、树、图等等。

术语“NoSQL”最早出现在上个世纪90年代末期，然而真正为大家所熟知则是在2009年中期。起初，它只是由 Carlo Strozzi 创建的一个小型开源数据库，将所有数据以 ASCII 文件的形式存储，并使用 shell 脚本而非 SQL 来访问这些数据。这个数据库与当前的“NoSQL”概念并没有任何相似之处。

Johan Oskarsson 在2009年6月于旧金山组织了一场会议，讨论 IT 市场的新技术、数据存储与处理等主题。之所以会举办这场会议的主要原因在于 BigTable 和 Dynamo 等新产品的出现。“NoSQL”这个术语则是由来自 RackSpace 的 Eric vans 提出的。这个术语原本就是用在这场会议当中的，也没有什么更深层次的含义。不过最后的结果却是它迅速在互联网

上蔓延开来，成为 IT 领域的一个新趋势。随后，Pramod J.Sadalage 与 Martin Fowler 编写了“NoSQL Distilled”一书，旨在对日益庞大的 NoSQL 世界进行组织。

现在大约有150多种 NoSQL 数据库（nosql-database.org），下面就来探讨一下 NoSQL 的主要发展方向。

列簇存储

面向列的 DBMS 是这样一种数据库管理系统，它将数据表存储为数据列而非行的形式。从物理上来说，表是列的集合，每一列从本质上来说都是只有一个字段的表。这些数据库通常用于分析系统、商业智能与分析型数据存储。

优点：

130 可以比较数据，因为在表的一列中，数据通常都是同种类型的。
131 可以通过便宜、性能一般的硬件实现高速的查询性能；由于压缩的原因，相对于关系型数据库来说，这种方式磁盘上的数据所占据的空间要少5到10倍。

缺点：

132 通常没有事务。
133 对于熟悉传统 RDBMS 的开发者来说存在不少限制。

典型代表：

134 HBase
135 Cassandra
136 Accumulo
137 Amazon SimpleDB

键值存储

你可以通过这种数据库将键值对存储到持久化存储中，随后使用键来读取值。那么对于这种初看起来用途非常有限的解决方案来说有哪些好处呢？在根据键来保存/读取值时，系统是

非常高效的，因为它没有 SQL 处理器、索引系统以及分析系统等诸多限制。这种解决方案提供了最高效的性能，代价最低的实现以及可伸缩性。

优点：

- RDBMS 太慢了，SQL 游标的负担过于沉重。
- 采用 RDBMS 的解决方案来存储少量数据的代价有些大。
- 没必要使用 SQL 查询、索引、触发器、存储过程、临时表、表单以及视图等等。
- 由于其轻量级的设计，键值数据库可以很容易实现可伸缩性以及高性能。

缺点：

- 关系型数据库的限制可以从底层就确保数据的完整性，而键值存储就没有这些限制，数据的完整性是由应用来控制的。在这种情况下，数据的完整性可能会由于应用代码的错误而做一些妥协。
- 在 RDBMS 中，如果模型设计良好，那么数据库的逻辑结构就能完全反映出存储数据的结构，并且与应用的结构有所不同（数据是独立于应用的）。对于键值存储来说，要想取得这种效果是非常困难的事情。

典型代表：

- Amazon DynamoDB
- Riak
- Redis
- LevelDB
- Scalaris
- MemcacheDB
- Kyoto Cabinet

文档存储

文档存储指的是用于存储、搜索与管理面向文档的信息（半结构化数据）的程序，其中心概念就是文档。具体的面向文档数据库的实现是不同的，不过总的来说，他们都会以各种标准化格式对数据（文档）进行封装与加密，主要格式有 XML、YAML、JSON、BSON、PDF 等等。

优点：

151 足够灵活的查询语言。
152 易于水平扩展。

缺点：

在很多时候原子性是得不到保障的。

典型代表：

153 MongoDB
154 Couchbase
155 CouchDB
156 RethinkDB

图型数据库

图型数据库指的是使用图结构的数据库，通过结点、边与属性来表示和存储数据。根据定义，图型数据库是一种提供了无需索引而彼此邻接的存储系统。这意味着每个元素都包含了直接指向邻接元素的指针，因此没必要再通过索引进行查找了。

优点：

157 对于关联数据集的查找速度更快。
158 可以很自然地扩展为更大的数据集，因为他们无需使用代价高昂的连接运算符。

缺点：

159 RDBMS 可以用在更为通用的场景下，图型数据库只适合类似于图的数据。

典型代表：

160 Neo4j
161 FlockDB
162 InfoGrid
163 OrientDB

多模数据库

这些数据库包含了多种数据库的特性。

有两种不同的产品分组可以认为是多模的：

164 支持多种数据模型和用例的多模数据库。 比如说，ArangoDB 宣称它拥有键值存储的好处，同时还提供了面向文档以及图型数据库的支持。

165 支持多种模式的通用目的的数据库。 比如说，Oracle 的 MySQL 5.6支持 SQL 方式的访问，也可以通过 Memcached API 实现键值访问。

典型代表：

166 ArangoDB
167 Aerospike
168 Datomic

对象数据库

数据库中的数据都建模为对象、属性、方法以及类。面向对象的数据库通常适合于需要高性能数据处理的应用，这种应用一般都有非常复杂的结构。

优点：

169 相比于关系元组来说，对象模型最适合于展现现实世界，对于复杂、多方位的对象来说尤为如此。
170 使用层次特性来组织数据。
171 访问数据时并不需要专门的查询语言，因为访问是直接面向对象的。然而，有时也是需要使用查询的。

缺点

172 在 RDBMS 中，由于表的创建、修改或是删除而导致的模式修改通常并不依赖于应用。在使用对象数据库的应用中，模式修改类通常意味着还要对与当前类关联的其他应用类进行修改。这会导致对整个系统进行修改。

173 对象数据库通常会通过单独的 API 与特定的语言绑定，只有通过该 API 才能查询数据。在这方面，RDBMS 就做得很好，这要归功于它所使用的通用查询语言。

典型代表：

174 VelocityDB
175 Objectivity

176 ZODB
177 Siaqodb
178 EyeDB

多维数据库

这是针对在线分析处理的一种数据库，它可以从各种关系型数据库中检索数据，并且以某种方式将信息组织为类别和段当中。

典型代表：

179 GlobalsDB
180 Intersystems Cache
181 SciDB
182 Rasdaman

多值数据库

多维数据库的变种。主要的特性是支持使用属性来存储值的列表。

典型代表：

183 Rocket U2
184 OpenInsight
185 Reality

总结

NoSQL 的发展速度异常迅猛，不过这并不意味着关系型数据库就没落了。他们还会在很多场景下发挥着巨大的作用，并且与 NoSQL 数据库共存。我们现在处在多种持久化存储共存的时代，并不存在处于垄断地位的关系型数据库与 NoSQL 数据库。架构师们对数据库的选择将会基于数据存储本身的特性，以及所预估的数据量。

原文 http://www.infoq.com/cn/news/2013/11/introducing-nosql

那些应对 APT 攻击的最新技术

摘要：网络安全，尤其是 Internet 互联网安全正在面临前所未有的挑战，这主要就是来自于有组织、有特定目标、持续时间极长的新型攻击和威胁，国际上有的称之为 APT(Advanced Persistent Threat，高级持续性威胁)攻击，或者称之为“针对特定目标的攻...

网络安全，尤其是 Internet 互联网安全正在面临前所未有的挑战，这主要就是来自于有组织、有特定目标、持续时间极长的新型攻击和威胁，国际上有的称之为 APT(Advanced Persistent Threat，高级持续性威胁)攻击，或者称之为“针对特定目标的攻击”。这些攻击统称为新型威胁。



一般认为，APT 攻击就是一类特定的攻击，为了获取某个组织甚至是国家的重要信息，有针对性地进行的一系列攻击行为的整个过程。APT 攻击利用了多种攻击手段，包括各种最先进的手段和社会工程学方法，一步一步的获取进入组织内部的权限。APT 往往利用组织内部的人员作为攻击跳板。有时候，攻击者会针对被攻击对象编写专门的攻击程序，而非使用一些通用的攻击代码。此外，APT 攻击具有持续性，甚至长达数年。这种持续体现在攻击者不断尝试各种攻击手段，以及在渗透到网络内部后长期蛰伏，不断收集各种信息，直到收集到重要情报。

对于这些单位而言，尽管已经部署了相对完备的纵深安全防御体系，可能既包括针对某个安全威胁的安全设备，也包括了将各种单一安全设备整合起来的管理平台，而防御体系也可能已经涵盖了事前、事中和事后等各个阶段。但是，这样的防御体系仍然难以有效防止来自互联网的入侵和攻击，以及信息窃取等新型威胁。

新型威胁的综合分析

APT 攻击主要呈现以下技术特点：

1、 攻击者的诱骗手段往往采用恶意网站，用钓鱼的方式诱使目标上钩。而企业和组织目前的安全防御体系中对于恶意网站的识别能力还不够，缺乏权威、全面的恶意网址库，对于内部员工访问恶意网站的行为无法及时发现;

2、 攻击者也经常采用恶意邮件的方式攻击受害者，并且这些邮件都被包装成合法的发件人。而企业和组织现有的邮件过滤系统大部分就是基于垃圾邮件地址库的，显然，这些合法邮件不在其列。再者，邮件附件中隐含的恶意代码往往都是0day 漏洞，传统的邮件内容分析也难以奏效;

3、 还有一些攻击是直接通过对目标公网网站的 SQL 注入方式实现的。很多企业和组织的网站在防范 SQL 注入攻击方面缺乏防范;

4、 初始的网络渗透往往使用利用0day 漏洞的恶意代码，而企业和组织目前的安全防御/检测设备无法识别这些0day 漏洞攻击;

5、 在攻击者控制受害机器的过程中，往往使用 SSL 连接，导致现有的大部分内容检测系统无法分析传输的内容，同时也缺乏对于可疑连接的分析能力;

6、 攻击者在持续不断获取受害企业和组织网络中的重要数据的时候，一定会向外部传输数据，这些数据往往都是压缩、加密的，没有明显的指纹特征，这导致现有绝大部分基于特征库匹配的检测系统都失效了;

7、 还有的企业部署了内网审计系统、日志分析系统，甚至是安管平台，但是这些更高级的系统主要是从内控与合规的角度来分析事件，而没有真正形成对外部入侵的综合分析。

由于知识库的缺乏，客户无法从多个角度综合分析安全事件，无法从攻击行为的角度进行整合，发现攻击路径。

因此，在 APT 这样的新型威胁面前，大部分企业和组织的安全防御体系都失灵了。保障网络安全亟需全新的思路和技术。

新型威胁的最新技术发展动向

新型威胁自身也在不断发展进化，以适应新的安全监测、检测与防御技术带来的挑战。以下简要分析新型威胁采取的一些新技术。

精准钓鱼。精准钓鱼是一种精确制导的钓鱼式攻击，比普通的定向钓鱼(spear phishing)更聚焦，只有在被攻击者名单中的人才会看到这个钓鱼网页，其他人看到的则是404 error。也就是说，如果你不在名单之列，看不到钓鱼网页。如此一来，一方面攻击的精准度更高，另一方面也更加保密，安全专家更难进行追踪。

高级隐遁技术。高级隐遁技术是一种通过伪装和/或修饰网络攻击以躲避信息安全系统的检测和阻止的手段。高级隐遁技术是一系列规避安全检测的技术的统称，可以分为网络隐遁和主机隐遁，而网络隐遁又包括协议组合、字符变换、通讯加密、0day 漏洞利用等技术。

沙箱逃避。新型的恶意代码设计越来越精巧，想方设法逃避沙箱技术的检测。例如有的恶意代码只有在用户鼠标移动的时候才会被执行，从而使得很多自动化执行的沙箱没法检测到可疑行为。有的恶意代码会设法欺骗虚拟机，以逃避用虚拟机方式来执行的沙箱。

新型威胁的应对之策

一、总体思路

2012年8月，RSA 发布了著名的报告——《当 APT 成为主流》。报告提及了现在组织和企业中现有的安全防护体系存在一些缺陷，导致很难识别 APT 攻击。现有的防护体系包括 FW、AV、IDS/IPS、SIEM/SOC、CERT 和组织结构，以及工作流程等等都存在不足。

报告指出，应对 APT 需要采取一种与以往不同的信息安全策略及方法。该方法更加注重对核心资产的保护，以数据为中心，从检测威胁的角度去分析日志，注重攻击模式的发现和描述，从情报分析的高度来分析威胁。

报告提出了7条建议：

1、 进行高级情报收集与分析。让情报成为战略的基石。

2、 建立智能监测机制。知道要寻找什么，并建立信息安全与网络监控机制，以寻找所要寻找之物。

3、 重新分配访问控制权。控制特权用户的访问。

4、 认真开展有实效的用户培训。培训用户以识别社会工程攻击，并迫使用户承担保证企业信息安全的个人责任。

5、 管理高管预期。确保最高管理层认识到，抗击高级持续性攻击的本质是与数字军备竞赛战斗。

6、 重新设计 IT 架构。从扁平式网络转变为分隔式网络，使攻击者难以在网络中四处游荡，从而难以发现最宝贵的信息。

7、 参与情报交换。分享信息安全威胁情报，利用其他企业积累的知识。

Verizon 发布的《2013年数据破坏调查报告》中则更加简明扼要的概括了应对 APT 的最高原则——知己，更要知彼，强调真正的主动安全是料敌先机，核心就是对安全威胁情报的分析与分享。

二、技术手段分析

从具体的技术层面来说，为了应对 APT 攻击，新的技术也是层出不穷。

从监测和检测的角度，为了识别 APT，可以从 APT 攻击的各个环节进行突破，任一环节能够识别即可断开整个链条。

根据 APT 攻击过程，我们可以从防范钓鱼攻击、识别邮件中的恶意代码、识别主机上的恶意代码、识别僵尸网络(C&C)通讯、监测网络数据渗出等多个环节入手。

而不论从哪个环节入手，都主要涉及以下几类新型技术手段：

基于沙箱的恶意代码检测技术。要检测恶意代码，最具挑战性的就是利用0day 漏洞的恶意代码。因为是0day，就意味着没有特征，传统的恶意代码检测技术就此失效。沙箱技术通俗的讲就是构造一个模拟的执行环境，让可疑文件在这个模拟环境中运行起来，通过可疑文件触发的外在行为来判定是否是恶意代码。

沙箱技术的模拟环境可以是真实的模拟环境，也可以是一个虚拟的模拟环境。而虚拟的模拟环境可以通过虚拟机技术来构建，或者通过一个特制程序来虚拟。

基于异常的流量检测技术。传统的 IDS 都是基于特征(签名)的技术去进行 DPI 分析，有的也用到了一些简单 DFI 分析技术。面对新型威胁，DFI 技术的应用需要进一步深化。基于 Flow，出现了一种基于异常的流量检测技术，通过建立流量行为轮廓和学习模型来识别流量异常，进而识别0day 攻击、C&C 通讯，以及信息渗出。本质上，这是一种基于统计学和机器学习的技术。

全包捕获与分析技术。应对 APT 攻击，需要做好最坏的打算。万一没有识别出攻击并遭受了损失了怎么办?对于某些情况，我们需要全包捕获及分析技术(FPI)。借助天量的存储

空间和大数据分析(BDA)方法，FPI 能够抓取网络中的特定场合下的全量数据报文并存储起来，进行历史分析或者准实时分析。通过内建的高效索引机制及相关算法，协助分析师剖丝抽茧，定位问题。

信誉技术。信誉技术早已存在，在面对新型威胁的时候，它可以助其他检测技术一臂之力。无论是 WEB URL 信誉库、文件 MD5码库、僵尸网络地址库，还是威胁情报库，都是检测新型威胁的有力武器。而信誉技术的关键在于信誉库的构建，这需要一个强有力的技术团队来维护。

综合分析技术。所谓综合分析，就是在前述所有技术之上的，并且涵盖传统检测技术之上的，一个横向贯穿的分析。我们已经知道 APT 攻击是一个过程，是一个组合，如果能够将 APT 攻击最多环节的信息综合到一起，有助于确认一个 APT 攻击行为。综合分析技术要能够从零散的攻击事件背后透视出真正的持续攻击行为，包括组合攻击检测技术、大时间跨度的攻击行为分析技术、态势分析技术、情境分析技术，等等。

人的技能。最后，要实现对新型攻击的防范，除了上述新的监测/检测技术之外，还需要依靠强有力的专业分析服务做支撑，通过专家团队和他们的最佳实践，不断充实安全知识库，进行即时的可疑代码分析、渗透测试、漏洞验证，等等。安全专家的技能永远是任何技术都无法完全替代的。

原文 http://www.youxia.org/yingdui-apt-xin-jishu/

[Lean Saas] MVP 的 M

最近幾年，興起 Lean Startup 的風潮。主要的訴求是創業者應該先把重點放在打造一個 Minimum Viable Product (MVP)，觀察市場的需求，再行調整變化商業主軸成長。

說起來簡單，但沒人敢篤定的說，這個 Minimum 的 M 到底要做到什麼程度才足夠。討論一個點子，快速製造出一個原型，然後讓使用者快速反餽調整？這樣做的人很多，但卻不是人人 work。

更普遍的是你見到一個「草率」的產品上線，然後從此就快速沉寂消聲匿跡。

M 不代表草率

很多創業者在製作產品的過程中，常常不自覺得把「MVP」這三個字拆開來看，亦即只完成了「M」，以為實作了「Mininum」的產品規格，就可推出觀看市場反應。事實上，市場上有人會要的東西是 MV （Minimum Viable）。如果你的產品並沒有解決了任何有價值有意義的問題，只是一堆精美的介面功能，那麼是沒有人會買單的。只是浪費了時間浪費了金錢，生出一堆沒有用的程式碼。

相反地，若提供的解決辦法對某些人很重要，那麼就算產品介面再怎麼 crappy，他們還是有興趣停下來試用，甚至付錢給你希望開發者將把它做得更加完善。

M 的 scope

回到正題。我們終歸要討論 M 的 scope 到底在哪裡。

我的建議是：

不會寫程式的作法

如果你不會寫程式的話。先去找一個「會用」的工具湊出一個勉強能夠動的流程，解決問題，然後開始找人試用。

如果你找不到現成的工具可以兜，那麼還有一個方式。你可以拿非常仿真的 Mockup 工具，如 Axure XP、Balsamiq，把「可以解決問題」的流程和出現的畫面畫出來。

不要覺得這樣的步驟很麻煩，因為到這裡為止，還是只有梳理工作流程而已。

你可以從這樣的流程，粗略的知道後面將要面臨多麻煩的事。以想要開一個商店來說，你就會知道商店：

需要一個上架介面
需要傳圖介面
需要結帳介面
提供哪些結帳選項
提供哪些寄送方式，如何選擇
要用哪種條列方式管理使用者訂單

會覺得製作這些畫面很花時間對吧？這些都只是畫面而已。要寫出後面的程式可能是畫一張畫面「五倍」甚至是「十倍」的時間。

透過把畫面與流程畫出來的方式，可以讓你(很清楚的)知道要製作這個東西要花多久的「成本」（不管是時間上還是金錢上）。

不要因為覺得很麻煩，就使用「功能列表條列」、或者「單張畫面，列出滿滿按鈕與分頁去表示所需功能」的方式去估算工作時間。

因為事實上，在列表上，一個功能你都只會以為需要一小時或一天就能完成。而真實會發生的情形是，一條功能至少要開發 7 天。

當你一旦理解到要執行這個計畫所需成本可能是多麼巨大的時候，自然就會毫不猶豫的把不需要的功能砍掉，只專注在「必要的解決手段」。就這個例子，甚至最可能的情形，是上去

Shopify 上面開一個網路商店，或者是註冊一個 Wufoo (線上表單服務） payment integration Plan。

會寫程式的作法

會寫程式的作法反而相反。我相信各位開發者都很威，開發軟體的成本對你們來說都超低。但，請千萬千萬忍住，只做一個功能就好。思考哪個功能是「解決問題手段裡面絕對不可以沒有的」，只做這一個就好了。以 Logdown 這種部落格平台來說，標準就是只做一個編輯器，以及簡單的文章系統就好了。

（真的不要擔心作太少，一上線你就會知道就算只有「一個」也是會操掛你。相信我，實際試一次就知道了）

因為技術創業者都是「很會寫程式」才會出來創業的人。所以往往的毛病就是不打 idea 草稿，邊寫邊想。又以 Ruby on Rails 開發者（哈，在說我自己）最容易有這壞毛病，因為開發程式太快。所以 generate model, generate controller, 套個版，就覺得這東西可以上線，上線就等於可以開始收錢發財了。

然後，就沒有然後了。

作產品跟上班、接案做的東西驗收標準完全是不一樣的。上班、接案，你只要按照 issue tracking 上的「指示」完成工作就可以了。但是「產品」的「完成」標準，是「順利」的「幫 End User 解決他的問題」。而要做到這件事，產品需要花上大量的時間打磨：介面要做到讓用戶可以理解不會迷路。程式不會出現詭異的錯誤。按照 End User 的需求調整加強，甚至打掉重練。

一開始就作太多功能的問題是：開發者以為這些功能都已經「完成」了。但事實上，他們每一個對使用者來說幾乎都是「未完成」的。也許使用者可以從這麼多的功能列表「猜」出開

發者想要提供什麼服務，但是實際上他們不會繼續用下去。因為這個網站爛透了，每一個功能都是「未完成」。沒有解決到任何一丁點問題的網站，沒有人有興趣繼續用下去。

那麼你說，沒關係，我可以之後照使用者的反饋調整網站。但是問題又來了，做了那麼多功能「都有問題」，使用者的回報意見，自然涵蓋了所有板塊。那麼你該如何判斷出什麼問題沒做好才是最嚴重的呢？優先權該放在哪個模組呢？有些厲害的模組甚至是當初花了很多心血做的。結果使用者劣評如潮，你思考了下徘徊在投入心血改進和不如砍掉兩個選項。但狠的下心嗎？

然後，就沒有然後了。被第一次進來的 Customer Feedback **打爆，就沒有然後了。**

所以，只做一個能夠解決問題的功能。只做這樣就好了。Minimal "Viable" Product

> 不會寫程式：Write down User Story
> 不會寫程式：用手邊可以兜出的方案實作
> 會寫程式：只寫一個「可以解決問題」的功能就上線。

這是我的新書 Lean SaaS 的第一章。若你對接下來的內容有興趣的話，我將會在這本書繼續連載後面的內容（保證每週連載，約十期）。

連載期間價格 30 USD。**連載結束後** 45 USD。

本書最後附贈 SaaS **上線** checklist，**我會在書中的逐章解釋** CheckList **的每一條意義。**

歡迎各位讀者來信交流，我的 email **是** xdite@rocodev.com

原文 http://blog.xdite.net/posts/2013/11/19/lean-saas-mvp-m

NFS 挂了

由于我不在现场，从 log 以及跟工程师沟通中大概推断出出现问题的场景。client 开了若干个 rsync 进程同步，而同时又在进行大量文件的 NFS 挂载，导致 1G 的网卡全被打满，然后，就 NFS 就 hang 住了，表现就是 df 没有任何的反应，卸载不掉，挂载不上去。

首先是在 kern.log 里面发现了一坨类似如下的 log:

NFS: directory /tmp contains a readdir loop.Please contact your server vendor.

g 了不少文档，没发现几个靠谱的解释，这里有个看上去比较靠谱的解释，说是 2.6. 39 的 bug，不过我的 client 是 3.2.0-29-generic 的 12.04，server 是 2.6.32-38-server 的 10.04，并且按照链接的做法也无法重现，只得暂时作罢。

后来想想要不先 umount 掉挂载的 nfs 文件吧，umount 发现 "device is busy"，按照之前的做法，先通过 lsof 找到对应的进程然后直接杀了就好了，执行命令的事后才发现 lsof 以及 df 等了很久都没有结果，基本就是 hang 住了，后来想强制 umount -f，依然无效，最后发现有个 -l 参数可以使用，还有效了:

umount -l /mnt

然后就是在重新 mount -v 的时候出现了若干的问题:

mount.nfs: access denied by server while mounting

有种说法是使用的版本不对，换:

192.168.1.10:/shared/files /mnt/files nfs soft,intr,nfsvers=3 0 0

另外就是检查 server 上该起的进程有没有都起起来,portmapper, nlockmgr, nfs,

mountd:

rpcinfo -p localhost

对应的 client, portmapper, nlockmgr, status:

rpcinfo -p

然后还有说源端口小于 1024 的，需要在 exports 里面加上 insecure

这里还有一篇文档提到修 max_resvport, min_resvport。

这个解决了接下来又遇到了 hang 在下面这个 debug 信息的过程中:

mount.nfs: trying 1.1.1.1 prog xxxx vers 3 prot UDP port

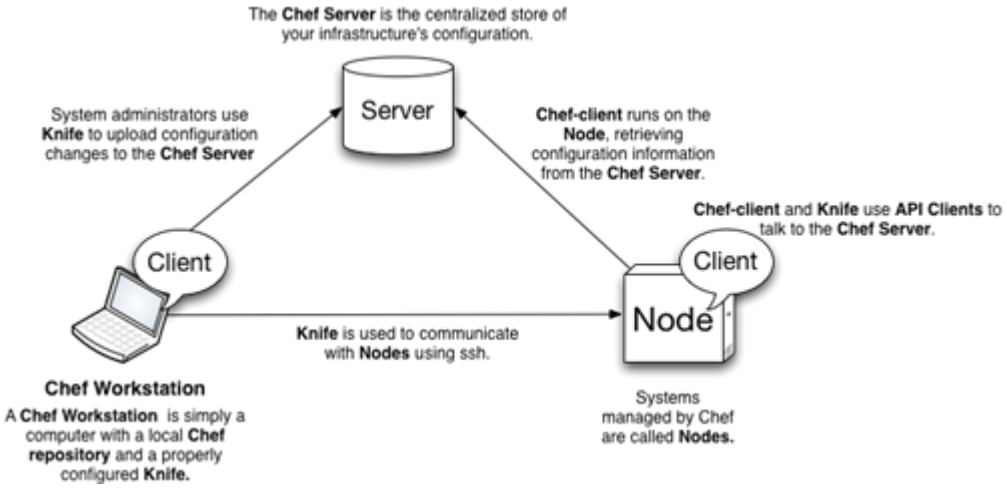
又是 g 了无数，最后在 gmane 的角落找到了一个临时的不是办法的解决办法，把版本改成 vers=2。

这里有个比较综合的 troubleshoting 的文档，说的倒是蛮全的，但是没有能解决上面的问题 :-

原文 http://jaseywang.me/2013/11/12/nfs-挂了/

Chef 的基本架构

Chef 是一个 C/S 架构的分布式软件配置管理工具。



Chef 架构

Chef Server 存储配置各个节点所需的所有信息，以 REST API 接口提供服务，Chef Nodes 是指那些安装了 chef-client 的节点，chef-client 主要用于与 Chef Server 通信，获取各自的 runlist 和 recipes，进行相应的软件安装、配置和启动工作。

Chef 通过 Cookbook 的方式进行软件配置，每一个 Chef Cookbook 定义了如何去配置某一个特定 service 或 app，整个配置过程可能被定义为多个 step，每一个 step 称为一个 recipe，Cookbook 需要上传到 Chef Server 由其进行统一管理。

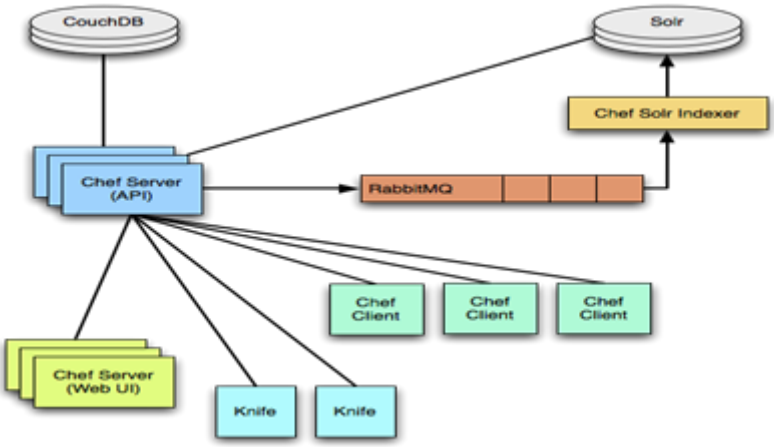
Chef 还需要为 Cookbook 定义相应的 role，一个 role 有一个定义 recipe 执行顺序的 runlist。每一个 Chef Client 可能需要安装若干个 role，运行时由 chef-client 进程与 ChefServer 通信，获得每一个 role 对应的 runlist，并根据 runlist 下载相应的 Cookbooks，依次执行各个 recipe 以实现某个 service/app 的安装配置。

在 recipe 中有一个关键部分叫做 resource，每一种 resource 定义一种特定的软件配置，譬如 rpm 的安装，用户/组的创建管理，目录的创建管理，service 的启动/停止等等，每一个 resource 对应一个 provider。开发 recipe 时可以按需求使用相应的 resource，chef-client 会由相应的 provider 来完成每一个 resource 的工作。关于 Chef 的 resource 可进一步参考 https://docs.opscode.com/resource.html。

关于 Chef 的基本概念和教程可进一步参考 https://learnchef.opscode.com。

Chef 的各个组件及其工作机制

Chef 的架构中主要包括 Chef Server, CouchDB, RabbitMQ, ChefExpander/Indexer, Solr, Chef Client, Knife 等。



Chef 组件（源自 https://wiki.opscode.com/display/chef10/Architecture）

Chef Server 使用 CouchDB 存储其所有信息，使用 Solr 提供 Search 功能，当有数据更新时，Chef Server 会更新 CouchDB 中对应的 document，并将数据通过 RabbitMQ 传递给 chef-expander，由其调度更新到 Solr 中。

当 Chef Server 接收到一个搜索请求时，Chef Server 先调用 Solr 的 Rest API 得到搜索结果，即所有符合请求条件的 document 的 ID，在根据该 ID 到 CouchDB 中获取每一个 document 全文返回给客户端。

Knife 是 Chef 提供的一个命令行工具，用于与 Chef Server 进行通信，Chef 的所有功能都可以通过 knife 来操作。

vSphere Big Data Extensions 是如何使用 Chef 的

在 vSphere Big Data Extensions（BDE）或 Serengeti 中，Chef Server 和 WorkStation 被安装在 Management Server 中，chef-client 被预装在 Serengeti 部署出的虚拟机中。Serengeti 安装好以后，在 Management Server 中已经配置好了 Chef Server，并为其配置好了 Serengeti 所开发的各种 roles 和相应的 cookbook，用户在创建 cluster 的时候需要为每个 nodegroup 指定相应的 roles，当虚拟机创建完成以后，Serengeti Web Service

收集虚拟机的硬件信息和用户定义好的 cluster 配置，通过 thriftservice 发送给 Ironfan，Ironfan 负责与 Chef Server 交互，在 ChefServer 上创建各个 Node 和 Role 信息，最后启动每个 node 上的 chef-client，通过 ChefServer 获取各自的 roles 和 runlist，并下载相应的 Cookbook 进行软件的安装配置。

关于 Serengeti，ChefServer，以及各个 Node 之间的拓扑关系，可以参考我们的另外一篇博客《**深入解析大数据虚拟化的架构（下）- 系统架构**》

实例讲解 Chef 和 Serengeti 的工作机制

在本部分中我们通过实例讲解 BDE/Serengeti 是如何通过 Chef 来安装软件的，为了降低复杂度，我们选用 Zookeeper 作为例子，因为 Zookeeper 的配置比较简单且对其他 Cookbook 的依赖性较小。本篇我们并不会详细讲解如何开发 Cookbook，相关的教程网上可以找到很多。

（1）准备 Cookbook 并定义相应的 role

Serengeti 的所有 Cookbooks 和 Roles 位于目录/opt/serengeti/cookbooks 中。

先看一下 Zookeeper 的 cookbook 代码结构：

```
$ treecookbooks/zookeeper/
cookbooks/zookeeper/
├──attributes
│  └── default.rb
├──metadata.rb
├──README.rdoc
├──recipes
```

```
|  ├── default.rb
|  ├── install_from_package.rb
|  └── install_from_tarball.rb
└──templates
└── default
├── java.env.erb
├── log4j.properties.erb
├── myid.erb
├── zkEnv.sh.erb
├── zoo.cfg.erb
└── zookeeper-server.erb
```

对于 zookeeper 这个 role 的定义：

```
$ catroles/zookeeper.rb
name 'zookeeper'
description'A role for running Apache zookeeper server'

run_list*%w[
zookeeper
]
```

该 run_list 只有一项，意思是说如果某一个 node 中含有 zookeeper 这个 role，那么这个 node 在安装 zookeeper 的时候需要运行 cookbooks/zookeeper/recipes/default.rb 这个 recipe。

default.rb 这个 recipe 主要做以下工作：

a) 安装 Java，mount disks, 这一步是通过调用其他 recipe 来实现的；

b) 创建必要的目录，用户，用户组等；

该步骤中我们使用如下 code 片段来创建用户：

```
user "zookeeper" do
group "zookeeper"
shell "/bin/bash"
password nil
supports :manage_home => false
action [:create, :manage]
end
```

该 code 使用了 Chef 的 **user** 这个 resource 来 create 一个名为”zookeeper”的用户名，参考 http://docs.opscode.com/chef/resources.html#user ,该用户名属于”zookeeper”组，使用/bin/bash 作为其 login shell，不指定 home 目录。

c) 安装软件包，根据 distro 的不同可能通过 tarball 或者 RPM 的方式来安装；

d) 将 templates 目录的配置文件拷贝到指定位置并做进一步的配置，启动 Zookeeper Service;

e) 更新状态信息，以通知其他依赖于 Zookeeper Service 的 Node。

(2) 上传 Cookbook 和 Role,配置 manifest 文件.

```
$ knifecookbook upload -a /opt/serengeti/cookbooks/cookbooks

$ kniferole from file /opt/serengeti/cookbooks/roles/*
```

查看 manifest 文件

```
$ cat/opt/serengeti/www/distros/manifest

[

{

"name" : "apache",

"vendor" : "Apache",

"version" : "1.2.1",

"hveSupported" : "true",

"packages" : [

...

{

"roles" :["zookeeper"],

"tarball" :"apache/1.2.1/zookeeper-3.4.5.tar.gz"

}

]

},

...

]
```

该 manifest 文件中定义了 apache1.2.1的 distro 中支持 zookeeper 这个 role，且指定了从哪里可以得到 zookeeper 的 tarball。

（3）创建包含 zookeeper 的 cluster

创建包含有 zookeeper role 的 specFile，可以参考 /opt/serengeti/samples/default_hbase_cluster.json

```
{
"nodeGroups" : [
{
"name" : "zookeeper",
"roles" : [
"zookeeper"
],
...
},
...
]
"configuration" : {
}
}
```

使用 Serengeti 的 CLI 创建 cluster:

```
serengeti>clustercreate --name test01
--specFile/opt/serengeti/samples/default_hbase_cluster.json --distro apache
```

Web Service 创建完所有虚拟机以后会通过 Thrift Service 调用 Ironfan 的 API，并将必要的虚拟机信息传给 Ironfan ,这些信息包括每个 node 的 rolelist ,ip 地址 ,位于哪个 physical host 上，配置了哪几个 disk，tarball 的完整下载地址以及用户自定义的配置信息等等，Ironfan 基于这些信息在 Chef Server 上创建各个 node 的信息，最后启动每个 node 上的 chef-client。

我们可以 login 到 node 上查看一些 chef-client 的信息：

```
$ cat/etc/chef/client.rb
log_level :info
log_location STDOUT
chef_server_url "http://<chef_server_ip>:4000"
validation_client_name"chef-validator"
node_name"test01-zookeeper-0"
```

手动启动 chef-client，从 log 信息中了解 chef-client 的运行状况

$ sudo chef-client -c/etc/chef/client.rb

关于 vSphere Big Data Extensions:

VMware vSphere Big Data Extensions（简称 BDE）基于 vSphere 平台支持大数据和 Apache Hadoop 作业。BDE 以开源 Serengeti 项目为基础，为企业级用户提供一系列整合的管理工具，通过在 vSphere 上虚拟化 Apache Hadoop，帮助用户在基础设施上实现

灵活、弹性、安全和快捷的大数据部署、运行和管理工作。了解更多关于 VMware vSphere Big Data Extensions 的信息，请参见 http://www.vmware.com/hadoop。

原文 http://vbigdata.blog.51cto.com/7526470/1325149

Cassandra 代替 Redis?

最近用 Cassandra 的又逐渐多了 ,除了之前的360案例 ,在月初的 QCon Shanghai 2013 篱笆网也介绍了其使用案例。而这篇百万用户时尚分享网站 feed 系统扩展实践文章则提到了 Fashiolista 和 Instagram 从 Redis 迁移到 Cassandra 的案例。考虑到到目前仍然有不少网友在讨论 Redis 的用法问题 ,Redis 是一个数据库、内存、还是 Key value store?以及 Redis 和 memcache 在实际场景的抉择问题，因此简单谈下相关区别。

首先，Redis 和 Cassandra 完全是适合不同的使用场景的 NoSQL 产品。前者是适用小规模内存型的 key value 或者 key list 数据，后者适合存储大规模数据。因此这篇文章提到切换主要原因或许还是前期 Redis 使用场景不合适，在初创公司项目初期，以顺手的工具快速实现功能的做法也可以理解。

Redis 的几种使用场景

186 访问量大

187 key value 或者 key list 数据结构

188 容量小，可控，可以全部放入内存。由于 Redis 是单线程设计，因此大 value 会导致后续的请求一定的堵塞。另外 hashset 当 hgetall 时候由于存在遍历操作，也不适合集合太大。如果数据超过单机容量可以使用常规的 sharding 方法分布到多台机

189 需持久化的场景

上面四点一般情况下应是必要条件。因此常见网站的用户资料、好友列表就适用用 Redis 来保存。由于 Redis 具有 memcached 所有的特性，也有讨论说 memcache 是否可以退出了？在以下情况下，我会倾向于选择 memcached 而非 redis

190 简单无需持久化的 key value，比如100字节以下。这种情况下使用 memcached 空间更节约且维护更简便。

191 有滚动过期需求，如网站的 session，每个新登录的用户定期过期。

相关观点也可参考 Memcached 真的过时了吗

几个问题

- 既然 Redis 可以持久化，用 Redis 保存的好友列表是否还需要保存到关系数据库？
- 手机游戏 Clash of Clans 中的城堡属性、及用户的金币、圣水、奖杯适用用什么数据结构保存？
- 原文 http://timyang.net/data/cassandra-vs-redis/

编程能力与编程年龄

程序员这个职业究竟可以干多少年，在中国这片神奇的土地上，很多人都说只能干到30岁，然后就需要转型，就像《程序员技术练级攻略》这篇文章很多人回复到这种玩法会玩死人的一样。我在很多面试中，问到应聘者未来的规划都能听到好些应聘都说程序员是个青春饭。因为，大多数程序员都认为，编程这个事只能干到30岁，最多35岁吧。每每我听到这样的言论，都让我感到相当的无语，大家都希望能像《21天速成 C++》那样速成，好多时候超级有想和他们争论的冲动，但后来想想算了，因为**你无法帮助那些只想呆在井底思维封闭而且想走捷径速成的人**。

今天，我们又来谈这个老话题，因为我看到一篇论文，但是也一定会有很多人都会找出各种理由来论证这篇论文的是错的，无所谓了，我把这篇文章送给那些和我一样准备为技术和编程执着和坚持的人。

论文

首先，我们先来看一篇论文《Is Programming Knowledge Related to Age?》，这篇论文是两个北卡罗莱纳州立大学计算机科学系的两个人 Patrick Morrison 和 Emerson

Murphy-Hill 对 StackOverflow.com 上的用户做了相关的数据挖掘得出来的一些数据。（我们知道 StackOverflow.com 上的数据是公开的，任何人都可以用来分析和统计，所以这篇论文的真实性是有的）

数据采样和清洗条件如下：（数据全量是1694981用户，平均年龄30.3岁）

192 15-70岁之间的用户（这年龄段的用户被称做“Working age”），当然，有很多用户没有输入年龄，这些用户都被过滤了。

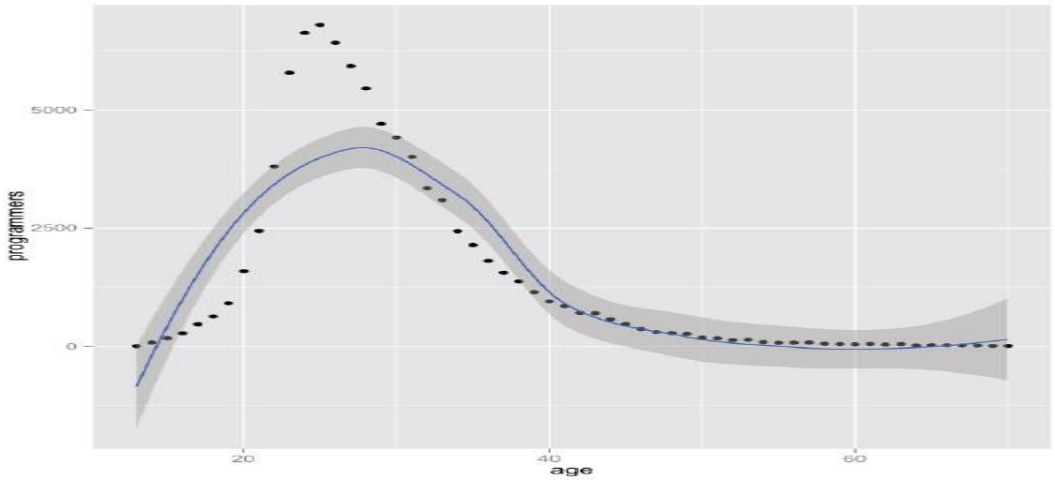
193 用户在2012年内都回答过问题。因为 StackOverflow 在2012年对问题和答案的质量要求得比以前高了一倍，所以更能反映程序员的真实水平。

194 Reputation 声望在2-100K 之间。（注：StackOverflow 的用户 Reputation 是得到社会认可的，在面试和招聘中是硬通货币。比大学的学分更有价值）

上述的条件一共过滤出84,248名程序员，平均年龄：29.02岁，平均 Reputaion 在1073.9分。

年龄分布图

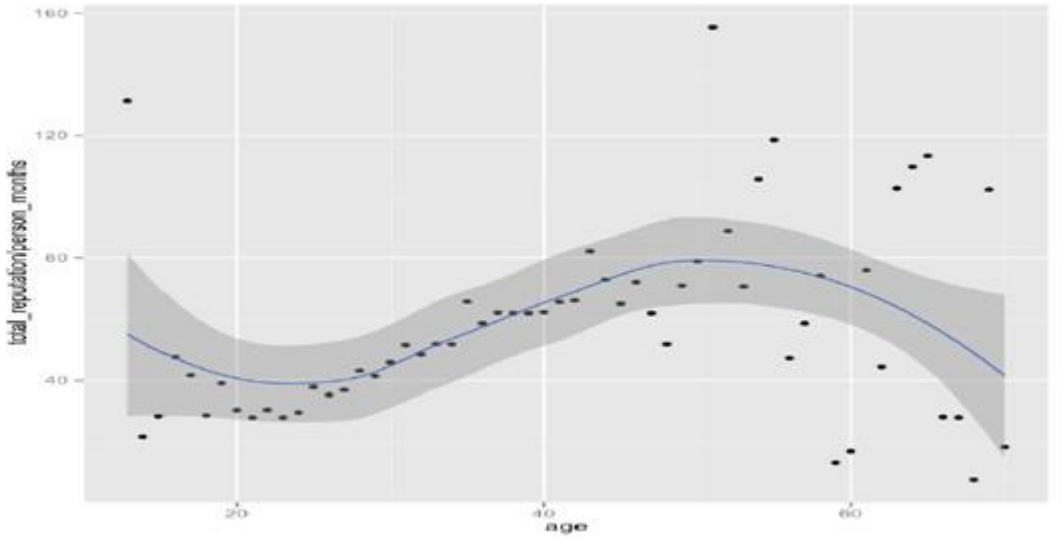
下面我们来看一下他们的年龄分布图：我们可以看到程序员年纪的正态分布（高点在25岁左右，但是中点在29岁左右）



能力和年龄分布图

然后，计算每个人每个月的 Reputation，这样可以找到这个用户的真正的活跃时间，这样便于计算这个程序员的真实能力。（总声望 / 活跃时间），可以得到他平均每个月得来的 Reputation。

我们来看看程序员的能力和年龄段的分布图：（你可能会大吃一惊）



上图中我们可以看到，程序员的能力在从25岁左右开始上升，一直到50岁后才会开始下降。所以说，程序员吃的不是青春饭。只有码农，靠蛮力，用体力而不是用脑力的程序员才是吃青春饭的人。

年纪大的人是否跟不上新技术

论文的作者分析了 Tag，用了最近5年内比较流行的技术 Tag，然后用了一套比较严谨的算法来查看那些所谓的“老程序员”是否在新技术上跟上不了，所谓跟不上，也就是这些老的程序员在回答这些新技术上并不活跃。所谓老，就是37岁以上的程序员（就是我现在的年纪）。

得到了下表：可以看到，老程序员和年轻的程序员对于一些新技术的学习来说也是差不多的，甚至有些项还超过了年轻的程序员。

Table1: Ten 'new' technologies and younger/older statistical comparisons

tag	t-value	p-value	mean-younger	sd-younger	n-younger	mean-older	sd-older	n-older
clojure	-0.85	0.802	3.25	3.58	539	2.95	2.88	84
django	0.02	0.49	0.91	2.01	4763	0.91	1.72	470
git	0.32	0.62	1.82	3.44	2852	1.78	2.23	426
ios	4.3	8.87E-06	0.96	3.19	19627	1.25	2.9	2119
jquery	0.36	0.358	0.78	2.02	28546	0.79	1.96	3016
linq	0.71	0.24	1.26	2.28	2366	1.34	2.24	431
mongodb	-0.47	0.68	0.95	1.69	2093	0.91	1.38	252
ruby-on-rails	0.52	0.3	0.89	2.45	11089	0.93	2.22	1061
silverlight	0.55	0.29	0.63	1.23	1487	0.69	1.48	235
windows-phone-7	3.41	0.0004	0.68	1.48	2158	1.16	1.68	153

结论

论文的结论是：

1）程序员技术能力上升是可以到50岁或60岁的。

2）老程序员在获取新技术上的能力并不比年轻的程序员差。

我的一些感受

最后，我说一说我的一些感受：

195 这些年来的对于外企和国内感受—— **国外牛 B 的 IT 公司的工程能力并不见得比国内的要强多少，但是国外那些 NB 的 IT 公司的架构和设计能力远远超过国内的公司，最可怕的是，那些有超强架构和设计能力的“老程序员们”还战斗在一线，这些战斗在一些的老鸟的能力绝对超过100个普能的新手。**

196 对年轻程序员的感受——国内新一代的程序员们太浮燥了。**老实说，对于大多数人来说，如果你没有编程到30岁，你还不能成为一个“合格”的程序员**。所以，并不是编程编到**30岁就玩完了，而是编程编到30岁才刚刚入门。**这些不合格的程序，整天 BS 这个不好，那个不好的，而且喜欢速成，好大喜功。

197 我是一个奔四的人了，编程就像登山一样，越往上爬人越少，所以，在我这个年纪还有想法，对编程还有热情的人不多了，基本上都是转 Manager 了。**其实，什么职位，Title 都是虚的，公司没了什么都没了，只有技术才是硬通货。而且，越是这个年纪还在**

玩编程玩技术的人，其实其经验和能力都是比较强的，都是中坚力量，如果还有其它这个年纪和我一样的人，求交往。

原文 http://coolshell.cn/articles/10688.html

也谈谈程序员职业规划的几个问题——我的一些故事

1. 职业发展的三个阶段

我将一个人的职业发展分为三个阶段：

（1）30岁以前

打基础，人生方向基本确定。30岁之后再做出变化，说不想做这个，会付出很大的代价。要改变自己的职业方向，30岁之前做出决定，越早越好。就像软件的需求变更一样。

（2）30-45岁

个人能力强、知识全面、经验丰富、性格成熟，适合带领团队，事业腾飞阶段，升职加薪。创业的最佳时期。

（3）45岁以上

人生定型，斗志渐退。当然不排除有人大器晚成，仍然保持旺盛的斗志，拥有过人精力和激情，像肯得基的创始人哈兰.山德士，65岁才创业成功。

以上表述并非像程序代码一样精确的东西，希望不会有人钻牛尖吧~ 还记得我曾说过的30岁现象吗？如果到了30岁左右，自己的个人能力还没有一个质的飞跃，危机会就慢慢降临。

2. 有目标才能成功

我有一位初中同学，由于中考成绩不佳，自费读了一所中专学校，学美术。20年后的一天，我在北京见到了他，今天的他已经成为了一家不大不小的装修设计公司的老板，成为众人眼中成功人士。当我们问他怎样取得这样不斐的成就时，他淡然的说，我初中毕业后，就在想一个问题，我将来要做什么，后来我给自己确定了一个目标，成为一个美术家，于是我去了美术学校读书。读中专时，我随老师接触到了一些室外广告设计的工作，我发现这个业务不但符合我的兴趣，而且有很好的发展前景，于是我稍微调整了一下我的发展目标。中专毕业后，我自学考进了中央美术学院，在那里，我接触到了一些跨国公司室内设计的业务，后来，为了跟老外打交道，我又到北京外国语学院学英语，在那里我认识了我现在的老婆。我能走到现在这一步，不是因我有多聪明，而是因为我一直知道自己想要什么，我一直有自己的目标。

他的这一段话，让我很震惊。想想我自己，我的高中生活在埋头苦读中渡过了；我的大学生活，在浑浑噩噩中过去了，我从来不知道、也没有想过自己想要什么，更加不知道什么叫人生目标，甚至大学所学的也是一个万精油的经济学专业。如果长此下去，我一生估计要给毁了。

后来一个偶然的原因，我转行做软件，给自己定了一个在当时简直高不可及的目标——当一名真正的程序员——这对于科班毕业的人来说，显得有点可笑。靠着大学期间一点微薄的计算机知识，我日以继夜的学习，终于慢慢达到并超过了自己的目标。后来随着自己的成长，我调整了自己的目标，考取了软件专业的硕士，获得了高级职称的证书，当上了项目经理，部门经理，公司副总经理。有一天，我对一位朋友说，我想写一本书，他笑了笑，没说什么，他有点不相信。一年后，我的新书《程序员第二步——从程序员到项目经理》终于问世了。这些小小的成绩虽然微不足道，但却见证了我的成长。在我心中，真正的成功，并不在于有没有创下一番伟大的事业，更不在于有没有赚到很多的钱，而在于我一直在努力战胜自己——因为我有了目标，我一定能做得到。

3. 听从内心的声音

一个人该怎样选择自己的职业的呢？自从我发表《从程序员到项目经理》系列博文之后，很多朋友向我问起这一点。

对于每个人，我都说同样一句话：听从自己的内心的声音。

这句话的出处是李开复的自传《世界因你不同》。当李开复每次需要做出重要的人生抉择时，他都会让自己静下来，听从自己内心的选择。正因为他每次都能找到自己人生的方向，李开复在每次人生的转变中，都获得了巨大的成长，一步步登上了人生的高峰。

听从内心的声音，是一种智慧，因为内在的直觉会帮我们抹去理智的灰尘，会告诉自己，我们到底喜欢什么，希望成为什么样的人。只要是内心做出的选择，永远都会是最正确的选择。如果每个选择都能经历一次，它未必是成就最大的那个选择，但一定是不会后悔的选择，这样就够了，不是吗？

我大学毕业后，在一家大型国企做外贸。有一天在听到朋友讨论职业发展时，我意识到在国企打发时间并非我所愿，做外贸所需要的人际交往能力也非我所长。那我要做什么呢？我心中涌现出三个字：程序员。我从来没有忘记我曾经对程序员职业充满向往，我喜欢那种用代码来控制电脑了快感。为什么我不可以做一名程序员呢？从那一天起，我才终于有了自己的职业目标——**当一名程序员。**

现在回想起来，如果我当初一直在国企，也许会很舒适安逸，但我一定会后悔，没有放手追求自己真正想做的事情。虽然一路走来，我并没有取得太大的成就，但我过得很充实，而且一直在提升，一直在做自己所喜欢的工作，我永远不会后悔，这就足够了。

4. 你适合做程序员吗

程序员是一份很特别的职业，他们的工作是跟机器打交道，需要长时间待在电脑旁边，工作的成果是一堆神秘复杂的代码。要胜任程序员的工作，是非得有一些特质的，例如：

（1）能享受编程的乐趣。

（2）坐得住，对着电脑一整天也不会觉得累。

（3）有很多关于软件的想法想要去实现。

（4）逻辑思维强，做事情，或与人争辩时，都很有条理性。这一点非常重要，一个人的逻辑思维能力决定了他写出的代码的质量。

（5）喜欢改进自己的成果，精益求精。 虽然我主张项目不应追求完美，但一个人的学习进步必须是精益求精的。

（6）热爱学习，愿意接受、学习新的东西。

有一些朋友在看到我的《从程序员到项目经理》的博文后，问：“为什么一定要当项目经理？30岁以后同样可以当程序员！”这个说法是对的，但有一个前提，就是你必须拥有程序员的这些特质，这样的程序员是不会有什么瓶颈的。一个人的瓶颈，不在于他的工作、公司或者领导，而在于在自己。如果你自问觉得做不到上面这几点，那你就真该好好想想自己的未来了——听从自己内心的声音，重新出发。

写到这里，总有点意犹未尽的感觉，夜已深，该休息了。

原文 http://www.cnblogs.com/watsonyin/p/3418934.html

云风：一个编程的自由人

属于小伙伴们的年代

他不知道我是个小孩，他到我家里找老吴，我妈以为找我爸的，然后就把我爸从单位里面叫回来，说有一个老头找你。其实他是来给我送十块钱的软件注册费的。

198 你从什么时候开始编程的？

我从小学一年级开始接触电脑。80年代初，电视都没有普及，我爸喜欢编程序。家里就买了一台9寸的黑白电视，专门为了接电脑，因为以前电脑是没有显示器的。所以电视就没看电视节目看，反正也没什么节目可看。那台电脑 Z80的 CPU，只有16K 的内存，比 Apple II 都差远了。这台电脑基本上只能写一些很简单的程序。我爸喜欢，这是他的玩具。

我刚上学，我爸就会给我写一些很简单的游戏。他最早给我练习100以内的加减法的程序也是用游戏形式呈现的。我爸给我写一些小游戏玩，我估计写程序是他的爱好，他不会觉得写出来的游戏好玩，但是我觉得游戏也是极好玩的。那时候写高性能的游戏程序就需要使用汇

编。而电脑上没有什么开发工具，我爸当时都是手写的汇编，然后拿手册去翻译成机器码，再导到机器里边，当时用的磁带机，程序需要保存在磁带上面。在磁带封壳上用笔标注每段程序的位置，玩的时候进带到那个位置，再连接上电脑读出来。

我小时候喜欢看书，家里的书我都拿来看，无论看不看得懂，反正我觉得有字的就看。有几本写程序的书，我就拿来读，看起来写程序可以帮助我完成学校留下的作业。我妈当时在上成人大学，学校开了个编程的课。他们是晚上上课，白天上班。她看我有兴趣，于是就把我拎到课堂上去听课。当时课堂上就我一个小孩，其他都是大人。很多人可能只是想要个文凭，所以那个课程都不大用心听。但我有兴趣，很认真的听课。老师很高兴，认真批改我交的作业。

199 真正系统学编程是什么时候？

真正系统学习是读初一的时候。我们中学新开了一个机房，购了一批中华学习机，也就是苹果2在中国的山寨版。学校没有什么老师教，唯一的一个计算机老师是我爸的大学同学，他想让我爸去代课，于是我爸就成了我们初中的计算机老师。编程课程整个学校只开了一个班大概有20、30个人，一个星期2节课，就在学校机房里上课。

当时气氛很好，玩得好的那些小伙伴，他们大部分现在也在写程序。我们有了个小圈子，在一起大家会有共同话题。不知道哪天开始就想自己做游戏玩，因为各种各样的游戏都玩过了，觉得也没什么，自己也可以做。所谓环境就是有很多同龄的孩子一起都在学这个，大家会比较，会交换自己写的程序，会讨论一下哪里没有写好。没什么资料，有些复杂的问题大家就会去想怎么把它做好。

当时的工具比较缺，不像现在电脑上什么编程工具都没有。所以你可能得顺带连工具一起做，比如说在 Apple II 上没有好的汇编器，你就得用 BASIC 自己写一个。那时候也没有互联网，书店里面的书虽然没有现在这么丰富，和编程有关的也有很多。我们没钱把书全买下来，就挑着买，大家买不同的互相借阅。 小时候有没有自己做出一些成型的软件？

我上中学的时候，就开始做一个看电子书的软件。我先在拨号 BBS 上发。后来《电脑报》每年出合集光盘的时候把它收进去了。当时见过很多共享软件需要注册，我就学样做了个启动几分钟后会弹出来的注册框。说你需要注册才能继续使用，注册费10块钱人民币，并留了地址。结果还真有人给我寄钱。那是我第一次靠写程序赚钱。我用一个本把谁买过这个软件

记下来，用平信把注册码寄回去。互联网流行之前，比较流行干这种事情。软件几年间大概卖了100份，也有1000块钱。

这里还有一个有趣的故事。竟然有一个退休的老头，他真的按我在软件里留的地址找到我家里去了，他去到我家里说找老吴，我妈以为找我爸的。打电话把我爸从单位里面叫回家，说有一个老头找你。后来问来问去说是找我啊，我当时在长沙读书。他是来送十块钱的。

上大二的时候，我有几个师兄接了个单：广东佛山有一个图书馆要做一个软件。我跟我发小，中学的朋友就想在假期打份零工。听说师兄们准备8月份去，我们就打算7月份跑过去先去做了。等那些人来，就没他们什么事了。就这样，事先没联系，一放假我们就直接坐火车跑到佛山去了。

他们的需求很简单，图书馆购置了一台当时很少见的光盘刻录机和扫描仪，想靠这个赚些外快。利用这些把附近居民拍的照片扫描成电子版，做成电子相册。图书馆的馆长说，你们有两个人，如果觉得一个人做这个东西就够了的话，我再给你们点事情。所以我们又帮他们做了一个图书馆的管理系统，两个人各做各的，拿当时比较新潮的 Delphi 做了两个星期。管吃管住之外额外收了几百块钱。我觉得过程挺有意思的，虽然他们有点抠门，但至少比学校的老师好多了。

200 大学毕业之后干什么去了？

临毕业最后一年我已经在北京一家游戏公司做兼职了。认识了一些朋友，刚毕业的时候，打算做手机平台的游戏。诺基亚刚发布了市面上唯一的一款智能机，投资人觉得手机应用会是未来的一个趋势。我刚毕业的那会，一直在折腾这个事情，我们的切入点就是做一个手机游戏的引擎。我们想做一个开发平台，大家在上面可以做游戏。当时选的是塞班平台。

上大学的时候，没想过去拉投资什么的，就是被人带着玩，开始比较有意思，搞了几个月，兴趣就没有那么大了。我觉得那时候手机上做游戏，只能做贪吃蛇、连连看，这些游戏不符合我的游戏观，我喜欢玩复杂一点的游戏。

在要签投资合同的时候，我作为创始团队成员，需要保证在这个团队若干年。我发现我要是未来几年全做手机游戏，就做不了我想做的东西了。于是在签字之前，说我不干了。大家也挺理解的。

接下来被北京游戏圈的朋友叫去做 web 游戏，那家叫可乐吧的公司。干了半年后没什么兴趣，就离开北京回武汉了。

网易那些年

丁磊最后说，你走也没关系，在网易留一个宿舍，你可以住这里，没问题。我觉得我在网易干得很开心，在这里不受限制。我不觉得公司欠我什么，我也不欠公司什么。

201 怎么去网易的？

大学快毕业的时候，我曾经帮一家游戏公司做了点东西。他们有一款游戏用了我的游戏引擎，需要我在此基础上再写一个模块。他们问我要多少报酬，我没多想，就按照同学在外面做家教的行情算了一下，我说就按家教1个小时10块钱算就行了。最后给了几百块钱，关系不错。

后来他们公司被网易收购，网易启动了《大话西游》项目，听说我没在工作，于是就继续让我帮他们做东西。我在武汉家中写一些模块，当然再不能按10块钱时薪做了，当时一个月有4000块兼职工资吧。做了段时间，他们觉得沟通太麻烦。丁磊打电话给我说，你不如就来广州网易干，机票帮你买好了，公司也帮你在公司旁边租了房子，过了马路就到办公室。你安心每天来上班，不用打卡。我爸说，你这么整天窝在家里也不像话，我就飞去了广州。

202 在网易做的第一个项目是什么？

2001 年的网易人还不多，就几十个人吧，我感觉那里的人都挺好的，特亲切。跟游戏部门一开始没在一起上班，他们上他们的，我上我的，我只需要完整需求。《大话西游》做了一年多就完成了。当时网易对他们的进度要求挺紧，似乎在按进度走，但我自己觉得质量很有问题，只是抢时间把它做出来。理所当然的，完成以后问题特别多，服务器容易宕机，客户端容易崩溃。

当时也没想过网络游戏后来会怎么发展，网易应该是因为石器时代的火爆而收购这个团队的。但网易当时的思路都是传统游戏开发运营的那一套，比如要找代理商来卖游戏客户端，就像一个有网络功能的传统游戏一样。

203 《大话西游》第一版不太成功，你接着做《大话西游2》了吗？

刚到网易时，我没把自己看成是游戏团队成员，只是他们让我做什么，我觉得事情还有点意思，就用心把图形引擎一小块的事情做好。项目结束后，整个项目没做好，觉得很不爽。丁磊那时候也没有信心，好在网易靠短信的业务开始赚钱了。

我记得那段时间，公司附近特别多的酒吧，丁磊喜欢泡酒吧。我就住在旁边，他在酒吧就打电话把我叫出来，也没什么事情，就是喝酒聊天。我记得那时候他觉得游戏没做好，而网易的聊天室服务还挺热闹，他就想把大话西游第一版改成一个图形聊天室。我说这个不好，客户端太大，那时候带宽又小，不行。

我始终觉得很多东西我来做的话，做好也挺容易的。原来的客户端不稳定，我在家里闷头搞了两三个月，重新写了客户端的雏形。弄好之后给大家看，说其实把客户端做稳定也不难。丁磊安排了原来技术部门的叮当来负责游戏项目，叮当从网易技术部门抽调了当时在网易技术最好的几个程序，决定把项目重做一遍。然后我就认认真真开始做这个项目了，这就是《大话西游》第二版。

在 2002 年，我们其实花了很短的时间，就把它重写完了。游戏的大框架都在，只是前一版程序写得不好。新的版本完成以后不放心，内部测试了很久很久，直到技术上没什么问题，才推了出来。我当时的想法就是在程序这个方面我尽力了，比以前稳定，至少不会崩溃，也没有内存泄露。

204 你在《大话西游2》上完全实现自己技术上的想法了吗？

我的一些想法也实施了，因为以前客户端很容易崩，我没太想去读老的代码。只是感觉如果按照我的想法做，会稳定一些。也的确好了很多。以前没把项目当成自己的东西做，所以有想法，最多说说，不会用力去推。后来到这个项目，就觉得是自己的项目，尽量把它做好，很多想法之前就考虑过，当时也比较成熟了。

我大学头两年以及中学时代主要是用 C 语言做开发，到大四开始写 C++的程序。后来几年都对 C++ 很有热情，有新的项目开始也就有了新的实践想法的机会。C++ 里可以玩的东西很多，每接触到一些就会想着如何用到项目里去。当时对语言特别热情，还做过 C++的 T 恤，把 C++印到名片上，利用公司的资源开办了一个 C++ 讨论组邮件列表，邀请了国内很多玩 C++ 的程序员参加。

接下来网易开始《梦幻西游》的项目，我参于的一个很大的动力是觉得 C++可以换个玩法，能够把新学到的东西用进去，可以换个方法写程序。因为前面一个项目还是比较传统的 C++的用法，后来就在《梦幻西游》里面用了当时比较新潮的 C++技术。等到2004年《梦幻西游》项目开发结束后，我就逐渐对 C++没兴趣了。然后到2005年的时候，在上海开了一次 C++的会议，我被邀请去做一个演讲。那时候，我记得听了一些关于 C++语言编程的高级玩法，虽然之前自己也花了许多时间去研究，但突然就觉得这些只是一个智力游戏了。

205 那个时候你写了一本书吧？

04年是我最闲的时候，我写了一本书，到05年的时候，那个书出版，我也到杭州工作，开始忙了。从 2005 年我正式写博客，最初的想法挺简单：我的书出版了，里面有很多错误的地方，以及想法的变迁，我想利用网络平台，来更正我写过的东西，所以这个博客最早的作用是维护勘误表，接着把新的想法写在上面。

我从 1997 年开始维护了很多年个人网站，之前也经常写些东西，网站本来就有一些固定的读者。在书出版后，有很多人就因为那本书找到我的博客，读者就更多了。我发现博客有更好的交互性，坚持写下来，一直没有停，写到现在。

206 你怎么去到杭州的？

2005年的网易，我已经算是干的比较久的员工。一些老员工离开了。我当时的想法是既然项目做完，《大话西游》都有几十万在线，在国内算是最好的一个网络游戏，那么我对自己的工作有了个很好的交代。网易的待遇不错，第一笔的期权收益拿到，似乎日后的生活没有问题。就想离开重新做点别的。

没想好去哪，当时也有其它公司让我过去，第一个找我的是盛大。盛大的陈大年在我读大学时，我们就是网友。他说想搞个研究院，问我有没有兴趣。

我直接跟领导叮当（前网易 COO 詹钟晖，于2011年5月15日卸任网易首席运营官，离职前是网易管理团队重要成员。他于1999年10月加入网易公司，2001年4月加入在线游戏事业部。2006年5月，詹钟晖被任命为网易联合首席运营官，2009年3月，詹钟晖担任网易首席运营官，主要负责游戏业务。）说我想换个地方，换个环境生活。盛大的朋友邀请我过去。他就问，去盛大能做啥呢？在网易不能做吗？

当晚丁磊找我喝酒，说你觉得哪里环境比较好，我说杭州感觉不错，然后他就说不如我们在杭州开个分公司吧，我说好啊。丁磊是浙江人，在杭州开分公司的事情估计他也是想了很久，恰巧借我的口提了出来。这个事情网易董事会批的很快，没几天我就一个人到杭州折腾去了。

刚到杭州，我在西湖边租了一个小办公室。一开始的几个人是我在游戏圈子里的朋友，还有过去的同学。接下来就是自己招人，初期人手不够时我还兼职过前台，行政，出纳。

207 当时在杭州想做什么？

一开始就想做游戏引擎，除了游戏引擎，还想做一款跟市面上的网络游戏不一样的游戏。04、05年是整个网易感觉最好的时期。没有竞争对手，收入不错，自我感觉非常好。刚到杭州时，老板不急，我们也不急，耐着心做就行了，我们想做一个不一样的游戏。我不太喜欢 MMO 的枯燥模式，觉得很枯燥。2004年，《魔兽世界》在美国在 Beta 测试，我却玩得挺开心的。发现网络游戏也不是那么无趣。

我想做点好玩的游戏，但觉得需要一步一步的来。技术跟不上，就先把技术做好，然后再做好的游戏。我想法很单纯，只想做事。我们只要7、8个人就可以了，人多了也管不过来。后来网易开始把重心移到杭州，前期因为在杭州已经有了我们这个办公室，并注册了分公司，成立杭州研究院也就方便了很多。早期我们在杭州研究院的筹备工作上做了不少事情，但后来的杭研的实际业务和我那里也没有什么关系。

208 看起来工作有些琐碎，你一个人能兼顾这些工作和开发吗？

我的确有一些非技术的工作，比如说我需要做公司内部的技术评审。除了杭州的事情，也经常回广州帮广州的项目解决一些问题，频率大约半年一次吧。因为丁磊的用人风格就是信任他熟悉的人。如果有项目出现问题，他觉得谁能搞定，他就把这个人调过去帮帮忙，他对我技术方面解决难题的能力很信任。经常把我叫回去做点这个，做点那个。我也乐意去做。

开发进行了几年后，我们有几个主要的程序员觉得累了，换到别的部门去。每次有人离开，我都很苦恼，因为需要接手那些原本别人负责的模块。后来发现我自己接的事情越来越多，我光接那些别人做的东西，就没有什么精力做新的东西了。每走一个人，我把他们做的东西都看一遍，再接着做，很累。最终整个工作室可能只有我知道所有的数十万行代码都是怎么回事。

在开发上我们摇摆过很多次，又想把游戏做好，又想把引擎做好，结果两件事情都没有做好。技术上的东西也是做了又改，改了又做。一开始我们的想法就是什么都自己做，觉得如果用公开的东西，外挂就容易钻漏进来。比如，如果我们连 runtime 都自己写的话，可以做出很强的加密壳。我们还可以自己开发脚本语言，不仅可以贴合游戏的需求，还不容易被人分析。这些想法现在看来是很可笑的，但在头一年，我们的确是在做这些。估计是受公司大气氛的

影响，觉得不用着急，什么都可以慢慢来。做着做着就发现，软件大厦不是几个人就可以搭起来的，你不能说自己干就从头自己干，我们当时不仅做了 runtime，还做了执行代码的加载器，我们做得特别特别细，干了许多本该是操作系统做的琐碎工作。一年多以后才醒悟精力不应该放在这些上面。

209 最后你们有成型的产品吗？

我们最后的3D 引擎半成品没有对外公开。到2010年的时候，网易的情况已经不一样了，网易的竞争者在08、09年之后开始多起来。但是我们还是比较封闭，没有很快的思路转换，觉得什么事情慢慢踏踏实实做就可以了，也不用管市场需要什么，因为网易有很多赚钱的产品，也不缺我这一个。我当时觉得有些东西做出来，公司内部可以用，我在里面学到的东西，我可以跟公司内部去分享。那段时间做的挺多的就是到广州做内部的分享，给大家讲我们做的东西，网易的技术氛围一直很好。

在杭州开发游戏引擎期间，我们工作室出过一款完整的卡牌游戏，我自己很喜欢，但用户不多，喜欢的人都很喜欢，不喜欢的人似乎也很难吸引他们来玩。公司也不愿意花资源去推这种特别小众市场的游戏。

我在杭州做的东西没有特别成功的，但是有很多技术上的想法，以及对游戏的想法总在和整个公司的同事分享。3D 引擎这个东西也很难做，它是需要很多年积累的，现在自己做3D 引擎的人也比较少。我们尝试过和广州的小组合作开发游戏，但没有合作成功。最后我们觉得我们小组要想在公司活下去有发言权，还得自己做游戏。所以到2010年，就开始重新招人自己做游戏。但这个时候，公司的环境对我们的支持已经不如2005年。

因为只是想先让工作室生存下去，所以也没什么高追求，加上之前做的卡牌游戏不被市场接受，我们这次觉得做一个市场认可的游戏就可以了，不打算做什么惊世骇俗的东西。最终开发时间太紧。2010 年底这个游戏就叫停了。其实对于公司来说，确实应该停掉，公司已经有太多项目了，没有特别的理由保留我们这个并不特殊的项目。

210 有没有想过把这个3D 引擎项目开源？

到2010年工作室解散的时候，我想过这事，和叮当说的时候，他说他做不了主，所以我们就退一步在公司内部开源。但毕竟在网易顶多也就1000个程序员，形成不了开源软件良好发展的社区，我们在这个引擎上继续继续做的工作很多。我跟丁磊说，我们做的东西如果公司不用，不妨把它开源，让大家一起完善。当时我们工作室刚解散，丁磊觉得我心里有想法。他有话直说，说你花公司的钱，做了4年的东西。你没做成功，就觉得你要找回一定的价值。你觉得开源就可以把自己这4年做的东西的价值体现出来。我说我不是这样想的，我只是很单纯觉得技术发展依赖交流分享。这个也是我最后走的原因之一，我觉得和老板有些理念很难达成一致。丁磊是很独断的一个人，公司什么都是他说了算，他认可的东西就很认可，他不认可的东西别人跟他解释也没用。丁磊为人不错，网易整体也很好，尤其是技术氛围，但并不是事事让人满意。

211 那时候是不是人也走的比较多，他也比较茫然，看问题负面的东西比较多？

网易游戏在前两年离开的核心人员比较多。大话西游的核心团队突然就走掉了一大拨，当时丁磊心里不舒服，突然飞到杭州。我带他去我一个朋友开的小酒吧，喝了一晚上红酒听了一晚上抱怨。最后我那朋友一直在我背后催我说太晚了要打烊了。

那个时候经常被他叫过去在办公室闲聊。我记得有一次他跟我说，中国搞航天什么的，花这么多钱，太浪费了。我说我跟你看法不一样，然后他就一个劲的说这个没用没用，浪费钱。他就觉得搞航天不如养猪，他有他自己的一整套的道理，所以你是很难说服他的。

到2011年的夏天的时候我离开。丁磊最后说，你走了也没关系，在网易大楼里给你留一个宿舍，你想住就住在这里，没问题。我觉得我在网易干得很开心，在这里不受限制。我不觉得公司欠我什么，我也不欠公司什么。

212 现在网易还有你这种类似于“自由人”的角色吗？

网易的牛人很多，我这种闲人现在估计没有了。我在广州的时候，以及杭州工作室解散后，经常在各个部门，工作室跑来跑去，对跟我没关系的业务发表些意见。比如 Twitter 正火的时候，我就说我们要不要做一个，我有很多想法。有事没事，我就会找人问说你们部门是不是闲了，我有个点子不错，我们要不要搞一下。因为我自己是闲不下来，就算没事，我也会

到一些部门去问，你们最近有没有遇到什么问题，我们一起看一看。我在网易工作 10 年多，大部分时间就是一个人，不用管人，也没人管我。我和谈的来的同事说，最近有些好玩的东西，我们研究一下。如果碰上可以做的事情，可能就搞上一两个星期。

最近听老同事说现在网易跟以前有点不一样了，以前老员工在网易还是比较悠闲的，爱什么时候上班，就什么时候上班。比如我当年就经常下午去上班，半夜离开。现在好象不行了，不管你有没有资历。

自由的程序员

213 每个人都会有自己的挑战，对别人来说很简单，但是对你来说很难，其实你都可以一点一点来完成。 后来怎么想到自己出来创业的？

2011年，我离开网易的时候想的蛮简单，觉得不在网易可能更自由一点。比如我想把我写的东西开源就开源，而在网易是不行的。我就想让自己独立出来玩一玩，想做点事情也好，更随便一点。

叮当比我早几个月离开，一开始我就知道他们几个人在想创业的事情。他们最后才找到我。起初一起合伙的人有做美术的，有做市场的，而做技术的人没有确定。后来这一伙人好些人还是留在网易了，并没有走。

他们一直缺个做技术的合伙人，我单身没有负担，是创业很好的合伙人选。叮当跟我联系。

我本来是想先休息，不过又想想，其实创业的话，能自己说了算也挺好的。我在网易07、08年以后研究的东西，都是做网游的平台，如果我是一个人做着玩，缺乏个平台，绝对是没法继续搞这些东西的。一个产品需要有很多的用户，游戏这东西不可能一个人弄。若是组建起新团队，做这些东西，知识和技能便可以延续下来，这个事情也蛮好的。

214 你们拉了很多网易的同事出来吗？

我们不太想找网易的人，之前大话西游的团队离开我也看到丁磊挺受伤。我知道这个对他确实打击挺大，以至于后来公司管理风格都变了很多。所以我们出来不太想找网易的人，我觉得就我们几个核心足够了，我们市场有市场的人，做产品有产品的人，做技术我也很有信心。

215 那你现在公司也成立了两年了，那你觉得达到你的目的了吗？

不到两年，还算达到了。其实我跟叮当干也蛮多波折，最主要是我觉得他们一开始人太多。都是很牛的人，我觉得做事情不能这么多人开圆桌会议来决定事情，所以我当时只有一个要求，我说叮当我跟你干，我听你的，没问题。反正我一直是你下属早习惯了。但是以后做核心决策加上我不应超过3个人。他们也挺犹豫的，毕竟我是后来的。

最后的结果是我们三个人出资成立了现在这家公司，然后再以我们三个人的名义去找了风险投资。最后公司董事会有 8 个人。

216 新产品什么时候上线？

我们的 MMO 产品叫《斗罗大陆》，这原本是起点一部非常有名的网络小说。原计划是今年底上线，但是现在可能进度赶不上。最大的原因是我们代理了一家由成都一家公司开发的产品《狂刃》。代理的产品有很多技术问题，我们抽调了很多人员帮他们去解决问题。不光是技术，还有产品的设计，都有问题。

《狂刃》已经上线。当初开董事会我赞成做这个代理项目的主要原因是可以借机锻炼一下运营团队，况且我们原本也有计划做用户平台，付费支付系统等。把这些用户平台提前做好，我们自己产品上线也会顺利一点。因为游戏最怕的就是刚上线时各种不稳定因素。大话西游那个团队出去，第一个产品失败遇到的重大问题之一就是他们的用户平台没有经过检验，第一次砸的宣传费几百万，因为第一天用户的接入平台出问题，搞了8个小时玩家不能登陆。这等于一个 BUG 就浪费了几百万的市场费用。我们提前验证一下自己开发的用户平台，代理一个游戏是个很好的机会。《狂刃》的第一天也出了用户登陆故障的事故，但是我们10分钟就搞定了，应急速度很快。估计等我们自己的产品上线的时候，出现这种问题的机会就更少了，也有了应急的经验。

217 你好象在 skynet 上面花了好多心思吧？

skynet 是我们服务器的一套框架程序。最初想法是在2010年，我们正在筹备做新的游戏的过程产生了新的想法。由于游戏项目后来迅速停掉，想法也没有实践。到2011年出来了以后，反正要做新项目了，就想想该怎么做。写了半年后尝试做开源。因为一直都想把自己做的东西迅速的放到程序员社区里面，大家都可以用，然后得到反馈，我们再改进。我一直觉得这是可以提高我们在技术圈中的影响力的好方法，同时也可以监督我们把自己的东西质量提高。

最近有很长一段时间，我在手游上面做开发研究，所以也不是全部精力在做这个东西。iOS游戏开发方面，也是我们全部自己做的。不过手游项目没有开源是因为有很多东西做得太快，针对性的代码太多，还没整理好。

我们写的很多东西跟业务没有关系的都尽量开源了。我对现在的同事也是这么说，如果你觉得你的东西值得拿出来，我们都鼓励你去开源。开源有一个好处，你如果能把项目推上正轨，有很多人会为它出力。就像我们 skynet 出来以后，一开始都是我一个人在维护，慢慢的发现有一些别的公司的项目也在用。会给你提意见，会推一些代码，发现 BUG 也会告诉我，慢慢这个社区就起来了。

218 手游方面你们是不是可以借用一些现有的游戏框架，比如 Cocos2d？

也有人问我们为什么不用 Cocos2d，我曾考察过一些 iOS 上面的图形引擎。我从大学开始就做2D 游戏，包括后来做《大话西游》、《梦幻西游》。我对这个领域很熟悉，知道2D 图形核心部分没有太多工作。所以我觉得没有必要用别人的东西。如果要开发3D 游戏，我们会选一个3D 引擎的开源东西做，因为自己做过3D 引擎，知道这里面代码量特别大，从头做很难。如果想尽早出产品，没有必要把精力花在引擎上面。但是2D 的东西，我自己花了一个星期的时间，就把基础的东西做的差不多了，工作量对我来说就是两周的工作时间。如果去用现成了的2D 引擎，可能我也需要两周时间去熟悉。而自己写的引擎，可以更清楚如何优化。手机游戏性能优化很重要，有点像20年前的感觉。机器性能都很差，你必须把优化做好一点。这方面我觉得自己还是比较擅长的。有很多东西新鲜有趣，比如在手机上考虑怎么省电等等。如果用现成的东西，还会放弃这些有趣的过程，我选择了自己来做。

219 你刚刚一直在提你觉得好玩的游戏，还有你的游戏观，你觉得好玩的游戏应该是什么样的？

我们现在做的3D 网游是按照魔兽世界的模式在做，我们不敢说做的比它好，因为我自己也是魔兽世界的玩家，我觉得做得比它好很难，但是我们是走这种模式，庞大的世界观，丰富的情节，有很多很多的细节，然后让你觉得是一个世界。

单纯个人选择来说，我觉得游戏系统越复杂，越有挑战性，我就觉得越好。比如说我原来不怎么玩手机游戏，我觉得手机游戏太简单。过年吸引我玩手游的第一个游戏是COC（《部落战争》），我突然觉得挺好玩，有人玩这个游戏就觉得不花钱就玩不下去了。而我反而觉得不花钱才是乐趣，有挑战。我没花钱但玩得挺嗨的。

我玩游戏对图象声效没什么要求，只需要它系统好玩。我也玩一些用字符拼出来的抽象游戏，比如矮人要塞这样的，画面上只有一些符号，但它有一个很复杂的游戏机制。还有一些策略游戏，通常会选最高难度，要求一步一步的精心策划，不准犯一点错误。可能我会不停的失败，但是我觉得失败也是一种乐趣，只要最终能成功就挺开心。

我玩游戏的口味跟大多数普通的玩家可能真不一样，所以我们自己的游戏，我不提太多的意见，交给策划去做。而我喜欢的东西，市场可能不接受。

220 你在业余时间经常攀岩？玩了多少年了？

我04年开始玩攀岩，05年就停了，在2011年工作闲下来才捡回来。我的身体素质不太适合玩这个，因为我体力不好，力量不怎么样，个子太高，韧带又不太好。攀岩比较适合1米7几的人玩，亚洲人超过1米8，力量就不太跟的上。虽然攀岩运动在20出头是黄金年龄，但玩到60、70岁绝对没问题。这不是项跟人较劲的运动，只是不断的挑战自己而已。这个东西，自己觉得开心就好。不为什么，甚至不是为了锻炼身体，纯粹是喜欢。玩攀岩也不占用什么时间，每次1个小时，一周三次不过3个小时。当然偶尔会开车出去玩。花上一个周末在野外。

攀岩是一项孤独的运动，就是让你觉得定期有事情做，让你觉得有很多挑战要去完成，每个人都会有自己的挑战，对别人来说很简单，对你来说很难，你可以一点一点来完成。你可以去各个地方，一直爬下去。玩攀岩的圈子，除了大学生刚毕业的以外，我认识圈子里面坚持下来的人，就是玩了好多年的人，很多岩友都是三四十岁。我现在是34，至少10年内我不会觉得玩得吃力。去野外爬，你经常会见到五六十岁的人挂在岩壁上。它不是一个纯年轻人做的事情。

今天完成不了的线路，明天可能也完成不了，但是你坚持，过一段时间，突然开窍了，就会发现很简单。攀岩是用脑子玩的，有的东西说不清，要自己体会。突然之间，原来很难的东西变简单了，然后你就会找下一个目标，可以爬更难的线路。也不一定是需要越爬越难，还可以尝试各种风格的，不同的岩壁。

221 你作为程序员出来创业，有什么切身感受？要是其他码农有这样的机会的话，鼓励他们出来创业吗？

我很满意现在的情况。虽然我们公司还没有正收益，我们还在花钱。现在公司最好的一点是我们可以很好的分工合作，大家做各自擅长的事情。就像我现在写程序，不管人。做开发的阶段，我只要把我的事情做好就行了。每个人把擅长的东西做好就行了。如果想自己什么都做，那就会很累。我自己的负面经验就是在杭州干了几年，什么都管，觉得自己什么都能做。当时想，只要其他人能做的事情，肯学肯适应，你都能做。结果是你没有那么多精力。还是把自己喜欢做的事情做好。你不喜欢、不擅长的事情，有人却喜欢擅长，交给他们就好了。

组建创业公司，做事的规则很重要。我觉得与人合作，矛盾迟早会有的。一开始就应该把决策机制和遇到矛盾大家怎么来协调定好。把坏的事情事先想好。不能依靠朋友间的友情来处理问题。原文 http://blog.codingnow.com/2013/11/interview.html

COS 访谈第十三期：逄伟(eBay)

写在前面的话：前阵子统计之都发布了一系列数据分析领域专家的采访，从学界到业界，从新人到鼻祖。而后统计之都的主编向我约稿，我第一个想起来采访的人就是 Wilson。虽然在 eBay 不同的部门工作，但同属数据科学的大圈子有交集也有不同的见解。我一直很好奇 CS 或者 EE 出身、从事系统开发背景的人会在工作中感受到什么样的数据分析，而 Wilson 正是回答这个问题最好的人选。



Wilson 是 eBay 美国 Engineering, behavioral insights and science 总监。他于2006年加入 eBay，先后参与过 eBay 支付系统、搜索产品、行为数据产品的开发。在加入 eBay 之前他在 IBM 从事了5年多的架构师和资深顾问工作。从浙大读书时期的创业经历，到咨询公司、再到电子商务公司，我们希望他对于学术知识、职业发展、数据科学的理解和心路历程可以启发后来者，以期在这个领域做出更大的成就。

Liyun: 我注意到你大学期间的专业是 EE 那当时你应该主要专注在工程方面吧？除此之外，那时候你们学很多统计吗？

Wilson: 当时和大多数本科生差不多吧，接触过一两门课吧。

Liyun: 当时有这种数据科学（Data science）的概念吗？

Wilson: 那时候是94年，我们刚进大学的时候做的还是 DOS 下的编程，拿三四张那种很大的磁盘装个 windows 3.x 都是很兴奋的一件事情。那时候哪里有数据科学，大家都没有多少数据。那时候我们做一个股票分析软件，就是在小小的内存上一个点一个点画的。我的第一台电脑硬盘的大小也就是640M 左右，内存也就是16M 还是32M 的样子。几十兆或者几兆就是“大数据”了——一张磁盘装不下的就是很大的数据。

Liyun: 那现在回想一下，当时学的什么东西最好玩？假设你现在回到学校，会再学点什么东西呢？

Wilson: 我的专业是 EE 嘛。从那时到现在，我们接触了无数的编程语言无数的编程工具，有些你可能现在都没听说过（Liyun：你刚刚说的那些我好像就听说过 VB），这是基本的编程的方面。此外还有网络相关和 Unix 下的编程、多线程的编程等等。从开发的角度，接触了很多很多，从没有可视化到有可视化，从 DOS（字符界面）到单片机编程、再到 C++等可以一个个画表，再到 VB、FoxPro，把一个数据库可以很快的展现出去。我们自己做一些可视化的工具，用那么一点小小的内存和硬盘处理图形是比较困难的，有很多好玩的技术挑战；另一方面从数据库到企业信息管理系统，我们做过好多，比如杭州环保局的系统软件。（Liyun: 这是你上学的时候还是后面到了 IBM 之后？）这都是在读书的时候。那时候你可以很快的把别人文章里面的东西搬到电脑上去，可以赚很多钱（当时一个月三四千！）。这个过程其实也很好玩，虽然技术挑战不多，但是很快你能看到技术带来的生产力。而且你能看到，很多人从很原始工作方式（比如打电话、写一些表格然后相互传来传去），到后面有个系统可以电子办公了，这也会让你觉得比较好玩。

此外，如果说哪些是比较有用的，那从当时的利用有限的内存画图、到现在的大数据 Hadoop 之类，很多算法和数据结构的原理是没有区别的。比如我们现在在分布式系统下，做很多数据的 join，这些 Hadoop 系统现在做的还不是很好的地方。说到底，它用到了很多 hash join, merge sort 这些算法，当年我们做那些小程序和现在做这些很新潮很复杂的系统，本质上是没有区别的。我觉得现在很多 Hadoop 开发者都有点走偏了，其实他们可以写很多应用程序，但是他们不懂这些。还有一点真正有用的是，在做这些的过程中遇到的困难会给你带来解决问题、分析问题、设计系统、调程序的经验，这些经验现在还是一直沿用的。当年我去从客户的需求角度开发画图系统、和现在把 eBay 无数用户行为数据用可视化展现出来找到 insights，过程和思考的逻辑是没有区别的。

其实这些年有很多像你们这些统计、数据科学背景进入这个行业，我自己也在看一些相关的书籍。如果说我有一个师弟学计算机，不需要很多，他有四五门课（算法、统计、线性代数、概率论、数学分析）掌握的非常深的话，那我认为他就是前途无量。只是当时我在大学的时候没有这个行业、很多事情也看不清。

我碰到过很多人，一些是 eBay 的研究员（research fellow），一些是国内的做机器学习做的不错的人、知道很多模型、也知道怎么去编程实现。相比而言，前者会用很基础的思维在做，比如看一下分布、画直方图、做一些基本的数学变换什么的，看起来没那么神奇却很有效；而后者可能会用一堆复杂的模型、把数据都扔进去、生成很多的特征，比较各种出来的结果，虽然也能解决很多问题，但面对真正复杂的问题他们可能只能解决一半、深下去就无能为力了。

Liyun: 就是说他们不是“由底到上”这样一种方式对吧，只是在上面飘着。

Wilson: 对的，这时候就在考验你基本的数学的知识。回到前面的话题，如果你有很好的统计、线性代数、概率论等这些基础，也学过一些进阶的机器学习模型，那作为数据科学家来说你大概完成了3/5；还有少数的一些人除此之外在工程上也很强，比如分布式系统，就可以尝试结合理论和实践的力量，开发引起行业革命性变化的产品。到这一步就是比如 Google 的 Jeffrey Dean 这样大神级的人物，引领行业潮流。

Liyun: 还有1/5？

Wilson: 完美的话，除了上述个人能力之外，还需要一个真正的行业应用环境，来解决一些非常困难的问题并且得到酬劳，拿到各种资源的支持来解决事情。对于学生来说，最后一步在学校是不可能的，你只能准备好前面的基础之后在一个行业中真正施展。我没办法回到过去，但是现在的学生却是可以的。

Liyun: 那你觉得学校出来之后到 IBM 的五年，除了前面的知识积累和资源，其他还有什么比较珍贵的经验？

Wilson:这就涉及到第一份工作怎么选了。当时我们在学校兼职的做股票软件的公司已经拿到风投了，做的还是很大的，在陆家嘴期货大厦都有办公楼。所以我当时有了一些 start-up 的经历，虽然我不是创始人。其实后来刚工作的时候工资还没有我以前做 start-up 高。

那个时候在年轻人的心里 IBM 三个字母都是烫金的，在浙江省也就招了两个人，能过去自然感觉也不错。今天回头看，除了这个光环之外，我当时的选择还是很正确的。IBM 有一套很好的体系、无论是技术还是做事情、思维方式、待人接物、对复杂问题的处理，都有一套方法论。现在乍一听可能有点死板，什么东西都套用现成的方法论可能有点过了，但其实对于职场新人来说，经过这样的训练，哪怕到了新的环境新的公司，这些都已经是深入骨髓的做事方式了。这些我是受益非常大的。

Liyun: 那你当时学这些的时候痛苦吗？有没有什么血泪教训？

Wilson:当时我在 IBM 给上交所做券商的网上交易相关的系统，我们几个 IBM 的人带着一帮下包（全用 IBM 的人太贵了）。下包的质量难以保证，所以我们要做很多基础工作、架构这些，然后下包的人去填代码。那三个月我工作基本是从早晨九十点到凌晨两三点，项目上线之后我就直接去医院了。那时留下了慢性肠炎这种病，跟了我四五年，虽然后面养好了。有时候太投入了是件好事，很有激情去做一件事情，会学到很多得到很多；但如果让我回头看看这段经历，我是绝对不推荐的。人生是一场长跑嘛，给你带来的损伤可能是永远无法恢复的，身体的健康绝对不能作为职业发展的代价。

痛苦的话，像 IBM 这样乙方的公司，会有很多很多的客户让你觉得狗血。你要陪着笑脸跟他去讲，而他们可以随便发火、随便否定你。但他们是你的客户，做咨询的原则就是你要陪着笑脸、用非常专业的姿态去处理，要把事情做成。

我在 IBM 的五年每年都有跳槽的机会，每年都有一些让你心动的想法，收入可能比 IBM 高很多、也不会那么辛苦、不会那么痛苦。我坚持下来的原因，第一点是在一个公司一两年，你是学不到那个公司的精髓的。在一个公司，别人要信任你要认可你需要花一些时间、熟悉情况要花一些时间，把想法做出来又要花一些时间、做出来之后收益最大化又要花一些时间。没几年的时间你是做不出来的。你放弃了，很多事就只能做一半。第二点是任何公司都会有些不如人意的地方，有些政治味道。经历过前面的痛苦之后，你就会觉得这些没什么、你的心态会锻炼的非常非常好。如果这些都经历过了，你能非常系统、条理化的解决一些问题，那么你在大部分的情况下都可以这样系统的来解决问题。最后还有一点受益匪浅的，就是你在做咨询的时候会遇到各种各样的利益冲突。客户之间也有派系、不同业务部门，我们 IBM 公司内部也有不同部门，各种利益交织。在这个情况下，把一个事情做成，很多时候不仅仅是技术而更多是人的方面。IBM 教你把这些东西都放到台面上来，分析客户之间的利益、和你一起合作的人的利益在哪里，这样的训练会对你整个职业发展都有帮助。

Liyun: 那当时是什么原因 eBay 让你终究心动了一下？

Wilson: eBay 当时也挺有意思的。当时我们跟另外一个咨询公司给电信做项目，当时一个人的朋友是美国 eBay 的。eBay 要在中国建一个中心，问有没有推荐的人。当时我就在想，eBay 是个什么公司啊，我们这是 IBM 这是多牛的公司对吧？我们出去都是天天穿西装戴领带的。但做咨询就要天天出差，而我当时刚结婚，对于家庭来说，每两周回来一两天会是一个很大

的问题。另外一点是 IBM 也是个蛮官僚的体系。我当时的表现很卓越，也有很多人认可，但 IBM 说到底还是一个销售型公司。很多人走 fast track，给公司带来很多订单，而搞技术的人永远都没有这样的机会。同时 eBay 的情况是主管的职位而且不用怎么出差，我在 IBM 还不是主管。我当时就想着试试看吧，聊了一下觉得 eBay 做的事情和以前完全不一样。以前我们做的计费（billing）系统比 eBay 复杂多了，但 eBay 的用户量大很多、实时要求很高。聊下来觉得 eBay 的文化和环境都很不错，也是一个很全球化的公司。IBM 已经非常中国化了。就决定试一试。后来发现真的是一个非常正确的决定。

Liyun: 那你什么时候开始觉得数据分析带来超乎预料的效果？

Wilson: 大概是三四年前从计费开始做搜索。不仅仅是我从计费到做搜索，这也是 eBay 在经历一个很大的转型。eBay 以前是非常商务驱动的一个公司，我们招了很多 MBA 来做产品经理，写很多功能性建议（feature proposal）。我们作为工程团队，就是把这些功能一点不差的实现。然后 eBay 开始技术转型，我换到一个查询语义理解团队。我们开始做两个产品，第一个是收集用户的反馈、然后建立索引，也没有什么文本挖掘，就是可以打打标签，结果上线之后我们 CTO 每天早晨就是用这个产品。基本上我们一个功能上线后，一两天之内我们就能发现三四个漏洞，或者是增强某些功能的需求。当然后期我们也做了很多文本挖掘之类，这些都是后话。另外一个是众包，即利用互联网上的人改变搜索的结果，把他们的观点拿过来，改变 eBay 的站内搜索。这样 eBay 的搜索开始有了评价机制，每个新功能上线之前开始有反馈、知道怎么做改进。有很多这样数据驱动。这些都不是很复杂的系统，但非常有用。这些数据开始驱动用户体验。

接下来我们做了很多挖掘用户的行为数据、改进搜索算法。说到底，eBay 的搜索就是三个方面：买家、卖家、商品。对于买家来说，就是挖掘他们的需求，然后匹配卖家和商品的信息。我们知道的买家的信息就是搜索关键词和点击，而对于同一个关键字来说每个人的想法可能都是不一样的，我们需要用数据挖掘出不同人要的是什么。对于卖家来说，eBay 上有各种各样神奇的卖家，需要用他们的数据来改进算法。商品方面就更神奇了，什么东西在 eBay 上都找得到，有各种商品的信息和用户反馈。这些信息加入到搜索算法之后，带来的哪怕是1%的收入增加，绝对数字上就已经很夸张了。

学术上很多人谈大数据，大数据多重要、我在处理多大的数据量。很多人都是谈大小，而这些数据真正用起来哪怕一点点改进带来的都是钱，实实在在的钱。

第三个层面，我们做很多深入的分析。现在大数据领域，大家做的比较成熟的，第一块儿是产生很多报表、KPI 什么的。大部分公司都在做、都在用。第二块儿是一些复杂的数据和算法，数学科学这些年这么红火。还有一块儿大家刚开始做、整个行业做的都还不够。这就是介于中间的，我利用这些数据、怎么提高用户体验——比如 UI 怎么设计、按钮怎么摆放、该用什么样的颜色。我觉得这个是大数据在这里发挥余地很大。eBay 做了很多实验，比如增大图片的尺寸会带来收入上升，这些虽然不复杂但是对人机交互的贡献很大。举个例子，我跑到搜索的页面上，拉到页面底部都是下一页，交易转化率就会变得很差、基本就是一直在按下一页了。我们回过头来看页面，到了底部之后其实用户没的选择，搜索框看不到、只能在不同页面直接切换。做一个简单的变动、比如搜索框固定在顶部、用户可以直接去搜索另外的关键字，这样带来的交易转化就会多很多。类似于这样的分析，虽然不是复杂的算法，但是体现了大数据的强大。我们在做一些类似的行为分析，希望给大家带来一些耳目一新的产品。

现在其实已经不是谈大数据的影响的时候。如果你处在一个比较好的商业环境之中，你不做任何事情商业环境都会在增长，在浪潮之巅猪都能飞嘛。但如果在激烈的竞争中你希望更上一步，那大数据就是不可避免的、你必须做，只有做的好还是不好、不存在做不做的问题。

Liyun: 那你比较喜欢和什么样的人一起工作？

Wilson: 对我来说，我有几个条件。

第一个，就是你要有激情，我在中国工作这么多年，在美国也看到了很多人。一个感觉就是，一个人能不能做成一件事情，取决有有没有一团火在他心中燃烧着，他想做成事情。有些人不管处于职业发展的什么阶段、不管初入职场还是浸淫多年，但他非常想做成事情；还有一些人非常专业，会及时交付不会拖后腿，但我也要有我的生活、工作不是全部。对我来说，我倾向于前者，有成功渴求的人，我也很愿意帮助这些人成功。

第二点就是聪明。（打断：你指的是 IQ 还是 EQ?）我是喜欢找到聪明的人。你有激情，也非常聪明，那么第一是你想做事情、第二是你能做成。我受不了鸡同鸭讲那样。而有几个聪明人在一起，你说一半他就知道了，他说一半你也知道了，你说了一个主意马上激发了他另外的一个主意，这样的效率肯定不是乘以2的问题，而是指数增长。

第三点就是能符合文化，能一起把事情做成。当然怎么能通过面试发现这些特质就是另一个问题了。

Liyun: 那你现在的团队是很多元化的吗？

Wilson: 我喜欢多元化。可能大家来自不同的文化背景，有不同的脾气，但是跟以上三点不冲突。

Liyun: 你曾经和学术界很紧密的合作吗？过程中最大的收获是？

Wilson: 其实我本人并没有很深的参与进来，但是我的团队和学者们有很多紧密的合作。比如和郁彬、 David Goldberg。他是协同理论的奠基人之一，像 palm 的手写识别也是他做出来的。像郁彬的话你应该比我更熟悉（Liyun: 其实没有啦，我只是听了很多八卦…）。我能感觉到我从他们身上学到很多东西。任何复杂的问题，归根到底都是很基本的。他们和 career 中期做机器学习的人思路有很多不一样，他们可以把很多复杂问题简单化、回归基础理论并解决。比如统计学的学生看文章，他们是在学习一些基本的原理和思想呢，还是学习各种复杂的模型？当然第二点也是重要的，但是第一点是基础。这个和武侠小说很像。梁羽生就是一个典型的代表，每本武侠小说都有两类人：正派和邪派。所谓的正派武功呢，就是一开始你打基础的时间要长一点、见效慢一点，到了后期可以不断精进；邪派的武功呢，会有各种神奇的方法速成，很快我就可以跟人谈 SVM 等厉害的算法，解决一些问题，但后期就无法深入了。我觉得这些杰出的学者都归到正派，他们有各种对数学的理解，至于各种算法那只是实现工具、各种招式套路还是需要有内力做基础。

第二点就是说，他们都是对原则不妥协。很多时候做管理，大部分事情都要妥协来实现。美国的历史中有很多妥协，比如最近的美国政府关门开门、美国内战解放黑奴后成立联邦等。在公司里做项目，我这边的想法是一二三、另一方是四五，但不可能实现一二三四五，可能实现的是一二五，这一定是大家有所妥协。现实中我们都需要靠妥协来做事情。但这些人很奇怪，他们是不妥协的。

Liyun: 他们是相互之间不妥协还是联合起来对外不妥协？

Wilson: 我觉得他们是心中有一个标准。如果一个事情你做不到，他们是不会开始后面的。他们可能不擅长打交道、不擅长管理，但是后面事情真正做对、做大、做出影响，都源于他们这样的坚持。同时也需要我们这样的中间人，在他们不妥协的时候跟外界做出一些妥协，最终把事情做成。事情还是要做的。

第三点就是，这些人非常实干，动手能力强。我们有个 VP 级别的研究者，谈数学模型、写 R 代码什么的，可能比很多人要强许多。我想郁彬这些人也是一样的。他们会把事情落到实处、不会那么空。都是非常实实在在的。

第四点就是对细节的关注。大家都说细节决定成败，但我看到的是很多一线的管理者，都会说这是技术细节，让他们解决就可以了。我的感觉就是这些人对细节看的非常的细，会改变整个一个大的结果。说到底，我自己走技术管理路子，这些经验对我来说都非常重要。

Liyun: 你现在还是每天都在读很多书吗？

Wilson: 这是习惯吧。和健身一样，这是我养成的比较好的习惯。

Liyun: 推荐几本书？

Wilson: 说两本非技术的吧。一本是《平凡的世界》，路遥的。还有一本是《约翰・克利斯朵夫》，原型可能是贝多芬。这两本书影响了我个人的很多性格和信念。

《平凡的世界》对我影响比较大可能和我的经历有关系。我的小学和初中都是在农村长大的，这本书主人公从高中生就不放弃梦想，然后到城市去打拼、做建筑工人。但他很喜欢读书，坚持学一些文学哲学的东西，精神世界和现实世界可以分开。他周围的工友可能一起喝个酒啊、发个牢骚啊、唱个歌啊，甚至可能找几个女人。他可以选择那样的生活、或者做个生意变得有钱，他也没有。他就是非常追寻他的精神世界。他给我的激励就是，无论现实处在什么样的环境下，你都可以选择做自己想做的人、不向任何东西妥协。

我们村儿历史上有过一个大学生，我大概是第二个。我家里条件其实还可以，我有三个姐姐我是独子。但是毕竟是处于那种条件下，我转学转到县城高中，再到浙大，再到 IBM，再到 eBay，再到美国 eBay，经历了很多压力。从村庄到县城，发现周围的人见多识广；从县城到浙大，会遇到全国各地很优秀的人，带来不同的视野；到了 IBM，很多人在职业发展上走得很远，也给你带来不同角度；从中国来到美国，又是不同的环境，有些东西可能不起

作用了。很多人会经历不同的变化。我一直会觉得，你永远会在所在的环境中做的最好。有这么一个信念的话，你就一定会做到。再就是，有的时候现实会让人堕落。传统意义上的堕落也好，工作之后打打游戏消遣娱乐也是一种生活方式。但是如果你有一定的追求，那其他方面一定要付出更多。我也会从中吸取很多力量，激励我做别人不做的事情、远离很多事情。以前在咨询，陪客户吃喝玩乐什么的很容易，很多人也就走在这样的道路上。当然没有什么不好，但是我无法接受这样的生活。

《约翰·克利斯朵夫》如果你没看过，我会推荐你去看一下。我看过你最近的微博，写到你对于经济学的一些热情（Liyun: 那是很久以前写的被我偶尔扒出来了而已…），我很喜欢看到如此的热情。你去看约翰的一生、时时刻刻，都有这样的火在燃烧的。作者罗曼·罗兰写的非常浪漫，非常有激情，让我对生活的热情燃烧的非常非常厉害。一旦你觉得你对生活的激情不够的时候，你去读一下这本书。

至于我最近在看的，是一本旅居美国医生写的，《近距离看美国》系列，他会讲很多美国的历史、怎么成立这样的社会制度、美国的宪法的发展过程。所有的这些又是怎么和美国的现状结合在一起的。这本书让我对我现在的环境有了更好的了解。你现在看微博上大家谈社会制度和变革，这本书会让你比较系统的考虑什么样的社会制度是好的、好的制度是怎么来的。我太太最近刚信教，我也在看一些基督教的历史和文化。了解基督教的历史有助于更多的了解这个世界——政治制度、社会现象、欧美、以色列的战争是怎么来的。你会有一个很清楚的蓝图，不像以前觉得世界太大了。现在你会觉得有个脉络在里面。

Liyun: 说到这里我想起一本书，《新教主义与资本主义精神》，就是说基督教的发展对于整个欧洲西方社会思维方式和生活的影响。你可以抽空翻翻，很薄的一本小册子。

Wilson: 好呀。新教是一部分，旧约你也可以看看。为什么中东战争不断、以色列可以建国、以色列人在美国可以混的很好（犹太人自动获得以色列国籍）。这些都是比较好玩的一些书了。

Liyun: 最后的一个小问题，你最喜欢给你太太和女儿买什么？

Wilson: 给太太和女儿是不一样的。给太太有两类，一类是很实用的她想要的，你买的时候没有惊喜，甚至会把图片发过去确认，你们可能觉得有些老土，但对于老夫老妻来说也是很重要的。另一类一定是浪费、无用的。浪漫跟有用是扯不上关系的，而浪漫对老夫老妻也是重要的。有些时候是对她很重要、有些对她周围的人很重要。我送过一些项链、手表啊，我不确认她一定会喜欢，虽然她告诉我她很喜欢。有些鲜花啊什么的，大日子一定要有的，且一定是要别人看到的。

给我女儿就是逛街的时候看到玩具啊衣服啊，现在也开始买一些书什么的。这个年纪基本上给她买什么她都喜欢。

Liyun: 你觉得家庭对你的支撑主要是哪方面？

Wilson:我觉得两方面。我挺感激我太太的。我会经历各种变化，很多是意想不到的，也不是每个变化都很舒服。中间会有各种困难的时刻，你充满着野心去做一件事情，却突然间什么都变了。这个时候你需要怎么办？怎么去看清后面的一些机会？经历这些的时候我虽然很清楚一步步该怎么做，但总有沮丧、郁闷的情绪和时刻。我和我太太很好的一点是能超越现状看的更远一点，比如从人生的角度、整个公司的角度甚至社会的角度。最近我有些日子挺难过的，我前天和她打了一个电话、帮我分析一些问题。理智上这些分析我都是知道的，但有需要她帮我从情感中拖出来。我们两个除了当时的好感啊爱情啊，我觉得我们两个非常融

洽，认定对方是另一半。我们很清楚长远的目标是什么，我们互相会在对方陷入苦难的时候把他拉出来，走向更远的地方。这对我非常重要。

另外一点就是，说到底人一辈子可能活80岁也就差不多了，我现在差不多接近一半了。想想自己到那一天的时候，真正给这个世界留下了什么东西？我可以说我做了一些学问、研究、算法，当然确实是挺有成就的；我也可以通过努力和一些人一起对世界有一些正面的影响。另一方面，我人生的过程也算精彩，尝试了很多不同的事情，不是就那么匆匆而过。但还有一方面，那就是陪你一直在走的人是你的家庭。你的孩子是你留在世界上的。很多时候回头看看，工作一直在变化、事情一直在变化，但你的家庭是不变的、家人是陪你在一起的。很多时候，这是你力量的源泉。你怎么让这些人过得很幸福，就成为了你的责任。我不知道女孩子怎么想，但对于男人来说责任是很重要的。

原文 http://cos.name/2013/11/cos-interview-wilson/

享受造轮子的乐趣



在业内，由于技术布道师的传播，“重复造轮子”成了一个讳莫如深的话题，一个新的项目在各种讨论场合总是会小心翼翼的避开这方面讨论，一旦触及则会马上描述自己系统的独特性和必要性。富有经验的项目管理者，通常在开工前就会想好各种质疑的应对。但是造轮子也并非是一个雷区，有时候造下轮子也是无妨，不必遮遮掩掩，真的需要考虑清楚的是，轮子为了 KPI、晋级、虚有的声誉，还是真的出于兴趣及热爱？他们是否可以放在阳光下，和另外一个社区的产品从技术的角度去比较？

在 Linux 起步的阶段，在服务端有高端大气上档次的 Solaris，在桌面端有占据垄断地位的 Windows，如果抱着不造轮子的心态，也就无法让高高在上的 Unix 走下神坛，装进每个学生的学习的电脑，也不会成为最经典的开源协作的项目，在搜索引擎的领域也有同样的故事，一直到最近一两年，还有顽强的公司的不断进入这个领域。设计和使用一项技术就像厨师和美食家的区别，是不同层次的工作与体验。如果你的目标是给用户提供一个更好的操作系统，使用一个引来的产品很难使我们离目标更近。

在一方面，造轮子可以使我们加深对专业领域的认识，帮助我们重新思考这个领域；而且我们可以根据自己对场景的理解，思考改进原来的实现。更难的境界是 REINVENT，而不是迈出造轮子那一步。另外一方面，富有挑战的项目的轮子可以帮我们更好的聚集人才。好一点的公司都在谈人才争夺战，但是大量的公司对于优秀的人才并没有充分施展的空间。

Where Should Top Coders Work?也提到了类似问题，顶级的开发人才希望能发挥自己的才智解决业界最棘手的问题，因此在这些挑战的项目周围聚集优秀的人才之后，就有更好的机会通过创造项目中的智慧，发挥技术的杠杆作用去推动业务。

但是也得强调几句，同一厂中的重复造轮子不值得鼓励，技术团队在造轮子时候得把握好优先级及时机，在农闲阶段造轮子，农忙阶段发挥作用。

原文 http://timyang.net/architecture/enjoy-reinvent-wheel/

献给迷失的你——一名 IT 员工的职场心得

这些日子我一直在写一个实时操作系统内核，已有小成了，等写完我会全部公开，希望能够为国内 IT 的发展尽自己一份微薄的力量。最近看到很多学生朋友和我当年一样没有方向 ，所以把我的经历写出来与大家共勉，希望能给刚如行的朋友们一点点帮助。

一转眼我在 IT 行业学习工作已经七年多了，这期间我做过网页，写过 MIS、数据库，应用程序，做过通信软件、硬件驱动、协议栈，到现在做操作系统内核和 IC 相关开发，这中间走了很多弯路，也吃了不少苦。



我上的是一个三流的高校，就连同一个城市的人多数都不知道。因为学校不好也就没有指望能靠学校名气找一个好工作。所有的希望都寄托在自己的努力 上了，大一开学前的假期我就开始了学习。记得我买的第一本书是《计算机基础 DOS3.0》，大家别吓着了，其实当时已经普及了 DOS6.22了，只是我在 书店里看到了 DOS4.0/5.0/6.0的书，以为像英语那样是第四、五、六册，记得当时到处找 DOS1.0，现在想想也幸好我没有找到：）。开学前我 学完了 PASCAL，那时既没有计算机也没有人可以请教，我连程序是什么的概念都没有，只好死记硬背代码，然后拿纸写。我一直到大三才有了一台486，在 这之前用纸写了多少程序我也记不清楚了，只知道最长的一个我拿 A4大小的草稿纸写了30多页，我的 C 语言、C++ 、VC 都是在这样的条件下入门的。

所以说条件是可以克服的，希望我的经历多少给条件艰苦的同学们一点信心。第一次上机是在我姐夫的机房，我的心情激动的无与伦比，但是一上机我立 刻傻了眼，他们用的是英文版的 Win3.1，我的那点 DOS 知识都见了鬼，上机提心吊胆的一阵瞎摸，一不小心把 Word

弄成了全屏，怎么都还不了原，当时 真是心急如焚，我以机器被我弄坏了。第一个 C 语言程序，就是那个经典的 HelloWorld，我调了几个星期，上机机会非常少，也没有书告诉我开发环境 （TC2.0）需要设置，而且开始我都不知道有编译器，我甚至自作聪明把写好的程序扩展名从.c 改成.exe，结果可想而知。

大一学完了 C、X86的汇编、数据结构、C++ 。由于精力都花在自学上了，大一下四门课挂了彩，三类学校就是这点好，挂上一二十门也照样毕业。不过扯远点说，我那么刻苦都及不了格，可见我们国家的计算机教育有多死板。

大二准备学 VC 和 BC，当时难以取舍，后来选了 VC，不为别的，只为书店里两本书，VC 那本便宜6块钱。我的努力在班上无人能及，学的日夜不 分，大三有了计算机后更是如此，很多次父亲半夜教训我说我不要命了，我一直觉得自己基础差，记忆又不行，条件也不好 ，所以觉得只有多花点时间才能赶上别人。居然后来有许多朋友说我有学计算机的天赋，让我哭笑不得。我用的是486，16M 内存，1G 硬盘，当时同学们的配 置都是 P166MMX，我安装一个 Windows NT4.0需要一个通宵，编译一个 BC5.0向导生成的程序需要近两个小时，我的显示器是个二手的，辐射非常大，开机屏幕冒火花，看起来很酷的：）。有一 次程序写的太久，觉得怎么白色的编辑器背景变成了紫色，以为显示器坏了，后来才发现眼睛不行了，不过说来也奇怪，到今天我的视力还能保持1.5，真是个奇 迹。但是就是那台破机器陪伴我两年，让我学会了 VC、Delphi、SQL Server 等。后来那台机器给我阿姨打字用，据她说一天她正打的开心，一股青烟夹着火苗从显示器钻出来，之后它才寿终正寝。

大三假期找了个机会在一个计算机研究所实习，与其说实习不如说是做义工，工作了两个月一分钱没有拿。但是这两个月对我的发展帮助很大，让我早一 步了解了社会，刚去的时候我当然是一窍不通，在那里我熟悉了网络，学会了 Delphi 和 Oracle。由于工作很认真，得到了比较好的评价，在一位长者的 引荐下，我开始和他们一起做项目，这使我在大三大四

就有了自己的收入，大四又找了两家 MIS 公司兼职，虽然钱不多，但是在学生期间有1000多的收入我已 经非常满足了，我终于用自己赚的钱把计算机换了。



大四下开始找工作，这时我的工作经验已经比较多（当然现在想想非常幼稚），开始听父母的想去那个研究所，实习过那个部门也希望我能去，但是不知 道为什么最后不了了之，这种单位就是比较官僚，我一气之下就到了我兼职的一个公司做 MIS 的 Team Leader。在大三到毕业一年的时间，做过了各种 MIS，从煤气、烟厂、公安、铁路、饮食到高校，什么有钱做什么，工作也很辛苦，经常加班和熬通宵，从 跟客户谈需求到设计、编码、测试、交付都要上。那时觉得很有成就感，觉得自己还不错，现在想想真是很肤浅。

刚走上工作岗位的学生很容易被误导，各种开发工具让人眼花缭乱，同时也觉得很受公司器重，但这样工作永远是一个低层次的开发者。不要跟我说什么 系统分析有多么多么重要，多么多么难。你以为自己跟用户谈需求做设计就是系统分析和设计了吗，国内又有几个公司能够做的很到位很规范？我是 ISO9000 内审员，也在 Rational 公司受过多次培训，拿了4个证书，还有一个公司让我去做 CMM。这些我听过很多，但是很多事情到内就变了性质，一个公司不是 通过了 ISO9000或者 CMM 就能规范了，我现在在一家有几十年历史的外企工作，里面的管理不是一般国内企业能及的。作为一个毕业不久以前没有步入过社 会的学生，几乎不可能在很短的时间掌握系统分析和设计，面向对象、UML 只是一个工具，关键

是人本身的思想，不是说你熟悉了 C++ 、Rose 就能够做出好的设计，相反如果你具备了很高的素质，你可以用 C 写出比别人用 C++更加模块化的程序。

话说远一些，国内软件开发行业有一个怪圈，很多人觉得 VC > Delphi > VB，真是很搞笑。这几个软件我都做过开发，说白了他们都是工具，应该根据应用的需要选择采用哪个，而不是觉得哪个上层次。如果你因为用某个开发工具很有 面子而选择的话，只能说明你很浅薄。如果说层次，那么这些工具都不上层次，因为它们用来用去都是一些系统的 API，微软的朋友不会因为你记住他们多少个 API 或者多少个类就会觉得你很了不起，你永远只是他们的客户，他们看重的是你口袋里的银子。我也做过系统内核，我也封装过很多 API，同样我也不会看重 那些使用这些 API 做二次开发的客户，除非他能够作出自己独到的设计。

至于有人认为 C++ > C 那更是让人笑掉大牙，不妨你去打听一下，现在有几个操作系统内核是用 C++写的 ,又有几个实时系统用的是 C++ ,当然我也不是说 C++不好 ,但是目前的 内核和实时系统中 C++还无法与 C 匹敌 ,至于说 C++适合做应用系统的开发那是另外一回事。所以我的观点是不在于你用什么工具和语言，而在于你干什么工 作。你的设计体现了你的技术层次。

这样干了一年我觉得非常苦闷，做的大多数都是熟练工种的活，个人技术上没有太多的提高也看不到方向。所以决定离开这个城市去上海，寻求更好的发展，并且打算放弃我以前的 MIS 转到通信行业。

写到这里不能不提到我女朋友，我们是在来上海前半年认识的，她大四在我公司实习，公司派她给我写文档，我们的感情发展的很快。她告诉我很多事 情，她家原本是改革开放的第一批暴发户，她母亲爱打牌，输掉了几百万，还欠了很多债，她有男朋友，但是她对他没有感情，只因为他给了她母亲两万多块钱，后 来还强迫她写了四万块的借条，她男朋友背叛过她并且不止一次打她，现在逼她结婚，不然就要她还钱。这人居然还是一个高校的老师！她母亲把父亲给她的学费花 了，因为拖欠学费她没有办法拿到毕业证。

她母亲现在有病需要钱，我拿出了自己的一点积蓄并且跟朋友们借了一些，替她交了学费并给她母亲看病（后来才知道看病的钱又不知所终，就连她母亲 是不是有病我都不知道，但她也是没有办法）。这个时候我家知道了一些事情，坚决反对我和她在一起，她原来的男朋

友也极力破坏。无奈之下我们决定早一定离开 这个伤心的城市，并且瞒着我们家。由于时间仓促，我只准备了4000块钱，她仅有的几百块钱也被她母亲要去了，我买了三张票，一张是中午的，两张是晚上 的，中午我的家人把我送上船，他们一离开我就下了船，我和她乘坐晚上的船离开了这个我和她生活了很多年的城市，带走的只是一身债务。



没有来过上海的我们两个性倔强，都不愿意去麻烦同学和朋友。来到上海是傍晚6点半，我们都不知道该去哪里，我们找了一个20块钱的旅馆，这个房 间连窗户都没有，7月份的天气酷热难耐，房间里非常闷热。第二天我们开始租房子，因为身上的钱不多，我们基本都是步行，花了一个星期时间，不知道在浦东转 了多少圈后找到了一个400块的房子，但是我们都 不了解上海是付三压一，还要付半个月的中介费，买了一些锅碗瓢盆后，我们身上只有800块钱了。工作都还没有着落，这800块钱要支持到我们拿到第一个月 工资，为了省钱我们自己做饭，每天买菜只花两块钱，她非常喜欢吃（也可能她在大学经常挨饿的原因），看到她现在这样省吃俭用我真的很不忍心。

她以前的男朋友也没有放过她，经常打电话来骚扰，并且来上海看她，还说了不少恐吓她的话，她过于善良，说他以前毕竟帮助过她，叫我不要与他一般 见识。以后的每天在家就是苦等面试通知，原本我想迅速找一家 MIS 公司解决眼前的困难，但是她坚持让我不要放弃自己的理想，终于功夫不负有心人，我找到了 一家通信公司，4000块的工资虽然赶不上 MIS 公司给我开出的价位，但也够在上海生存。她也找到了工作，第一天上班她哭了，这是她来上海第一次流泪，我 心里很难受也很感动。

由于是全新的行业，我把自己降到了零点，我学的 VC、Delphi、数据库派不上用场，摆在我面前的是嵌入式、协议、信令一些我从未接触过的知 识。我知道我没有退路，于是拼命的学习，我把自己当做一个应届毕业生一样，一分努力一分收获，半年过去我终于熟悉了工作，并且得到了公司的表彰，薪水也加 了一级。后面的日子里我们省吃俭用，把欠朋友的1万多块钱还了，日子终于上了正轨。这时女朋友告诉我她想考研究生，我也很支持，于是她辞职在家备考。

另外，在这里我要感谢我的 Project Manager，他原来是一个大通信公司的产品经理，对人非常和善，我从他那里学到了很多知识，而且他也给了我许许多多无私的帮助。在工作上他给我充分的 空间和信任。记得公司安排我维护一个接入服务器软件，由于代码量不算太小（5万行），资料和文档都不齐全，我维护起来非常吃力，所以想重新把它做一遍，公 司领导不太支持，可能觉得工作量太大，但是他极力支持我，私下里他让我放手去做，我的维护工作他挤时间做。在他的支持下，我花了半年时间完成了接入服务器 的软件，并且实现了一个相对完整的 TCP/IP 协议栈。在这里我学会了嵌入式系统设计、驱动开发、TCP/IP 和很多通信的知识，我花了一年时间终于使自 己从 MIS 开发转到了通信行业，并且站稳了脚跟。我的开发大量是对硬件的直接操作，不再受微软的操作系统，VC、Delhpi 这些开发工具的约束，我终于 看到了另外一片天空。

我做事情喜欢追根问底，随着开发的深入，软件开发与硬件联系越来越紧密，硬件知识的匮乏又对我的发展产生了障碍，而且芯片技术基本上掌握在国外 公司的手里，这对做系统级设计是一个非常大的制约，一个新产品出来，第一道利润（也往往是最丰厚的利润）常常都被 IC 公司如 Intel、Motorola 赚去了，国内的厂商只能喝点汤。所以我决心解决自己的硬件技术障碍，并打算离开通信行业，进入 IC 设计相关领域。



当然我明白如果我对硬件了解的非常少，没有哪家 IC 公司会仁慈到招我这样一个一窍不通的人来培训。所以我必须努力打好基础，学一些相关知识为以 后做准备。就像我开始从 MIS 转到通信一样，我看过大量通信方面的书，并且给一个 ISP 做过 RADIUS 计费分拣台，在这样的背景下这家通信公司才给了我 这个机会。我在的通信公司是做系统设计的，有不少 PCB　Layout 硬件人员，平常我就注意向他们学习，由于我做的是软件，在公司看硬件资料不好意思，所以开始只好在家看，刚来上海工作我连续一年都在加班， 后来不加了，因为我要挤出时间学习，通常我12点左右睡，第二天5点半起，我上班比较早，地铁上如果人不多，我也用来看书。

学习当然不会是一帆风顺的，有些实在不懂的问题就积累起来问硬件人员，他们的帮助使我学习进度快了很多。因为在没有人点拨的情况下自学，我的一 半时间是花在解决疑难问题上，但这种问题经常是别人的一句话就可以让我豁然开朗，我非常庆幸我有这样的学习环境。在后面的一年里，我学会了看硬件原理图， 学会了简单的硬件设计 （模拟电路方面还有不小的差距），事情就是这样的，当你安安份份做软件，别人永远认为你是软件开发人员，在你开始学习硬件时别人未必会认同，有位中兴通讯 的朋友还对我说过，一个人不可能把所有东西都学完。



我也明白这一点，但我希望自己做的更好。但当你熟悉硬件后大家又会觉得你好像原本就是软硬件都懂的，同事们也都习以为常了。这个时候我可以把硬件资料堂堂正正的拿到公司看，没有人再大惊小怪了。让我比较自豪的是我通过自己的努力做了一个 IAD（软交换的终端设备）系统方案，包含软硬件的选型、设 计等内容，这个方案得到了公司和同事们的认同，让我感到非常欣慰。

技术是相辅相成的，当我的硬件有了一定的进步后，我的软件设计也有了很大的提高 ，我可以从更深层次理解问题。我做的接入服务器 CPU 是 Motorola PowerPC 860 熟悉的朋 友都知道860 QMC 与软件的批量数据传输通常采用 BD 表的方式，硬件人员做驱动的时候习惯采用固定 BD 表 ,每接收或发送数据都将数据从 BD 表拷贝到用户 Buffer，或从用户 Buffer 拷 贝到 BD 表。由于理解的比较深入，我自己重新实现了这个过程，采用动态 BD 表的方式，驱动从一个网口接收数据，提交给我的软件进行三层交换，直至从另外的 接口发送出去，没有进行一次拷贝。这样的设计大大提高了性能，使系统的指标接近理论值。

软硬件的结合使我的设计水平上了一个台阶。我现在写的这个操作系统，编译后我把程序反编译成汇编，找出其中不优化的代码，然后在 C 程序中进行调 整。举个例子，很多 CPU 没有专门的乘法指令，这个大家应该都知道，在这种 CPU 上进行一个乘法操作常常会花费大量的指令周期，有的朋友会说这个我知道。 我会尽量避免采用×号，但是事情往往不是那么简单，你知道 C 语言中数组的下标操作是怎么实现的吗？仔细看看反汇编的代码你就会明白，同样是通过下标的定位 操作，C 编译器会有时候会产生位移指令，但有时候会用

乘法实现，两者效率往往是天壤之别，所以明白这些问题你才能将系统性能提升到极致。这些问题就不多说 了，有兴趣的话以后可以共同探讨。

话说远一点，我由衷的希望在软件上做的比较深入的朋友们有机会学学硬件以及其它相关知识，尤其是做底层开发和嵌入式设计的。这对软件技术的提高 有非常大的帮助，否则很多事情你只知道该这样但不会明白为什么该这样。我这个观点在我现在的 IC 公司 Project Manager 那里也得到了验证。他告诉我们公司现在802.11芯片产品的软件经理原本是做该芯片硬件设计的，某某某原本是做软件的，现在在做 IC，类 似的例子还有很多，只是在国内这样的风气不是非常流行。

我有一些心得体会与大家分享，只有当我干好本职工作后，我才会学习与工作关系不大的技术，这样公司的上司才不至于反感。在入门 阶段的问题我通常不去问那些资深人士 ，而是问一些资历比较浅的朋友，比如刚毕业不久的学生，因为他们往往会跟你详细的讲解，而资深人士通常觉得你的问题太简单，所以回答的也很简单，我又不好 意思多问。

等技术上了一定的层次后我才会问他们，他们也能给你比较深入的回答。另外，有些朋友说我机会比较好，他们也希望能从事新的工作可惜没有机会，我 听了只有苦笑，我的机会了 解的人都应该知道，我没有出生在什么 IT 世家，也没有谁一路提拔我，所有的路都是自己走出来的，我母亲去世比较早，我的后母（我叫她阿姨）看着我努力过 来的，一次她看我大年30还在写程序，她说像我这样努力木头都能学出来。



一个好女朋友会帮助男人的成功

我的最终目的是 IC 而不是 PCB，所以我下一步的准备开始学习 IC 设计的知识。公司的同事没有懂 IC 设计的，后面的路又要靠自己了。我买了不少 相关的书，在网上也查了很多的

资料，我花了大量的时间去学习 VHDL，并且用软件进行了一些简单的设计和仿真（没有设计 ASIC，只是针对 FPGA）。随着学习的深入，我渐渐明白了 IC 设计的基本流程，同时也明白了这条路的艰辛。这个时候我已经做好了跳槽的准备，我向一 家业界又一定知名度的 IC 设计公司投了简历，并通过了漫长的面试（4个多小时）。其他的一切我都比较满意，唯独薪资差强人意，我也明白原因，因为我是这个 行业的新人，我没有经验，我再一次将自清零了。公司老板问我6000多一个月能不能接受，我知道他也是照章办事。想想我通信行业的朋友们，基本上都是年薪 10万以上，月薪过万的也比比皆是，朋友们也帮我介绍了不 少待遇不错的公司。我该怎么选择，当时我很犹豫，我热爱我的事业，我向往我的追求，但我也是一个普通的人，我也需要养家糊口，我也想早一点买房买车。生活 给我出了一道难题。

爱因斯坦在**63岁时说过一个人没有在30岁以前达成科学上的最大成就，那他永远也不会有。**这句话给了我很大 的压力和震动，我马上就26岁了，离30只有四年时间，我必须抓紧这几年宝贵的时间，努力达到我技术上的最高峰。为了这个理想，为了能离自己的梦更近一些，我选择了这家 IC 公司，我明白自己的薪资和公司刚进来的硕士研究生相差无几，但为了今后的发展只能忍受，一切又得重新开始。换行业是一个非常痛苦的过 程，尤其从一个春风得意的位置换到一个陌生的岗位，感觉象从温暖的被子里钻出来跳进冰水中，让人难以接受。在原来那家通信公司，我是唯一两年时间涨了五次 工资的员工，公司和同事都给了我极大的认可，工作上也常常被委以重任。但现在这一切都成了过去，在新的公司 我只是一个新人，没有人知道也没有人在意我过去的成绩。我决定重新开始，我把自己看作 新毕业的学生，我要用自己的努力得到公司的认可。进入新的行业是非常痛苦的，我告诉自己必须忍受这一切，虽然外面有很多诱惑，但是既然作出了选择我就不允 许自己轻易放弃。

我现在已经在这家新公司上了一个多月的班，开始非常艰难，现在慢慢适应了。第一个月结束时，Team Leader 找我谈话，说我是新进员工中最优秀的一个，我心里很欣慰，这也算对我努力的一个肯定吧。在这里还要感谢我的女朋友，她给了我很大的支持和鼓 舞， 每次在我动摇的时候她都在鼓励我 ,让我坚持自己的理想 ,刚来上海是她让我不要勉强去做 MIS，这次也是她让我顶住了月薪过万的诱惑 ,没有她我可能不会有今天的成绩。现在的公司有自己的操作系统，自己的 CPU、DSP 和其它芯片，在这里我能学到世界上最先进的技术，我们的设计开发不再完全依赖别人的硬件和系 统 ,这让我很开心。我打算等工作步入正轨后，全力学习新的知识，实现我的理想。



在后面的两年里我给自己定下了几个目标：

1、努力做好本职工作，在工作上得到公司和同事们的认同；

2、努力学习 IC 硬件设计知识，多向同事请教，并利用一切机会多实践；

3、实现我的实时操作系统的主要部分，完成 TCP/IP 协议栈模块，并免费发布源代码；

4、和我女朋友结婚并买一套小房子，这是最重要的，因为我明白事业是可以重来的，但是珍贵的感情很难失而复得。

在这里提一下我现在开发的操作系统，它是一个实时嵌入式系统，目前支持以下特性：

a. 支持时间片轮转调度和基于优先级调度，最多64个优先级；

b. 抢占式实时内核；

c. 为了便于移植，主体用标准 C 实现；

d. 汇编代码非常少，不到100行；

e. 支持任务管理，各任务有独立的堆栈；

f. 进程同步和通信目前完成了 Semaphore，Message Queue 正在调试；

g. 实现了定时系统调用；

h. 可以在 Windows 上仿真调试。

我还打算下一步实现优先级反转保护，Event Flag，Data Pipe，内存管理（以前实现过），驱动接口等。在这之后我还会努力完善它，比如加入文件系统，协议栈、调试接口等。希望朋友们提出自己的意见和建议，在此不胜感激！

原文链接 http://elevencitys.com/?p=5631